SRT Guida V2

A cura di Danilo Belvedere, Andrea Bisacchi, Davide Giordano e Fabrizio Torriano,
Revisione di Giacomo Romanini

PROBLEMA 1

In questo formulario avrete in **ROSSO** tutte le formule da applicare / calcoli da effettuare.
Troverete invece in **VIOLA** i risultati da inserire all'interno del testo che verrà consegnato.
NB: Tutti i test sono conclusivi quindi le risposte da dare sono Sì/Sì oppure No/No tranne per AUDSLEY (che può avere N/S oppure S/S) e DMPO dove segnato nella guida.
NB: Una qualsiasi risposta conclusiva di tipo S/S significa che i TEST successivi non sono da fare.

PREREQUISITI:

Vengono forniti dal testo un certo numero di processi Pi sottoforma di Tabella:

	T_i [t.u.]	C_i [t.u.]
P_1	10	2
P_2	15	4
P_3	20	5
P_4	30	3
P_5	60	2

- Inserire in tutti i casi la colonna **Ui (usata nel test di Han per esempio)** affianco alla colonna dei **Ci** calcolata come **Ci/Ti.**
- **Di** viene fornito dal testo, se non specificato **Di = Ti.**

Nell'esempio affianco U1 = 2/10 = 0.2

- Calcolare **Up** come la **somma di tutti gli Ui** calcolati precedentemente.
- Aggiungere la colonna **δi**, ogni valore è uguale a **Di/Ti**

a) Dimensionamento del ciclo MAGGIORE:

M = mcm (Ti) = mcm (10,15,20,30,60) = 60.

b) Dimensionamento del ciclo MINORE:

In questo punto bisogna restringere il numero possibile di m ad un insieme finito.
L'insieme di partenza da considerare è il seguente:
max(Ci)<= m <= min (Ti)
Nell'esempio avremo quindi **5 <= m <= 10** ovvero **m = {5,6,7,8,9,10}**

Restringere ulteriormente l'insieme degli m appena trovati applicando **M mod m = 0**

Tentare con tutti i numeri trovati nell'insieme precedente ed escludere i valori che non rispettano tale condizione. (Resto della divisione uguale a 0)
ES: **60 mod 5 OK** mentre **60 mod 7 NO**
m = {5,6,10}

L'ultima condizione da applicare per restringere m richiede il calcolo del MCD.
La formula da applicare per verificare se ognuno degli m trovati rispetta le condizioni è la seguente:
2*m - MCD(m,Ti) <= Ti ∀ i

In sintesi si effettua il MCD tra m considerato e **TUTTI i Ti** verificando che tale valore sia inferiore o uguale al Ti stesso considerato.

In caso di parità/più m disponibili, si sceglie quello più elevato.
CONSIGLIO: Partire da m più elevato possibile perchè al primo che becchi concorde con la formula puoi fermarti! (non troverai mai m migliori di esso).

Per calcolare il MCD occorre fare la scomposizione in fattori primi dei due numeri presi in esame e, dati i due insiemi della scomposizione, prendere come RISULTATO MCD il **prodotto** tra i valori **comuni** ai due insiemi con l'esponente più piccolo.
Es: 10 = 2*5 e 20 = 2^2 * 5 quindi MCD (10,20) = 2*5 = 10.

Es: m = 10 **2*m - MCD(m,Ti) <= Ti**

Alla prima iterazione di esempio avremo

2*m - MCD (10,Ti) <= 10

MCD tra: (10,10),(10,15),(10,20),(10,30),(10,60) e verificare se l'insieme dei valori risulta sempre <= del rispettivo Ti.

20 - MCD(10,10) <= 10	20 -10<=10	SI
20 - MCD(10,15)<= 15	20-5<=15	SI
20 - MCD(10,20)<= 20	20-10<=20	SI
20 - MCD(10,30)<= 30	20-10<=30	SI
20 - MCD (10,60)<=60	20-10<=60	SI

Evidenziare questo calcolo soprattutto se Ti non è multiplo di un m.
Avendo trovato come concorde alla formula il valore di m più elevato possiamo fermarci, dunque:

m = 10

c) Calcolo del numero dei cicli minori e dei Job per processo nell'ambito di ciascun ciclo maggiore:

ncm = M / m = 60/10 = 6

nj1 = M / T1 = 60/10 = 6
nj2 = M / T2 = 60/15 = 4
nj3 = M / T3 = 60/20 = 5
nj4 = M / T4 = 60/30 = 3
nj5 = M / T5 = 60/60 = 1

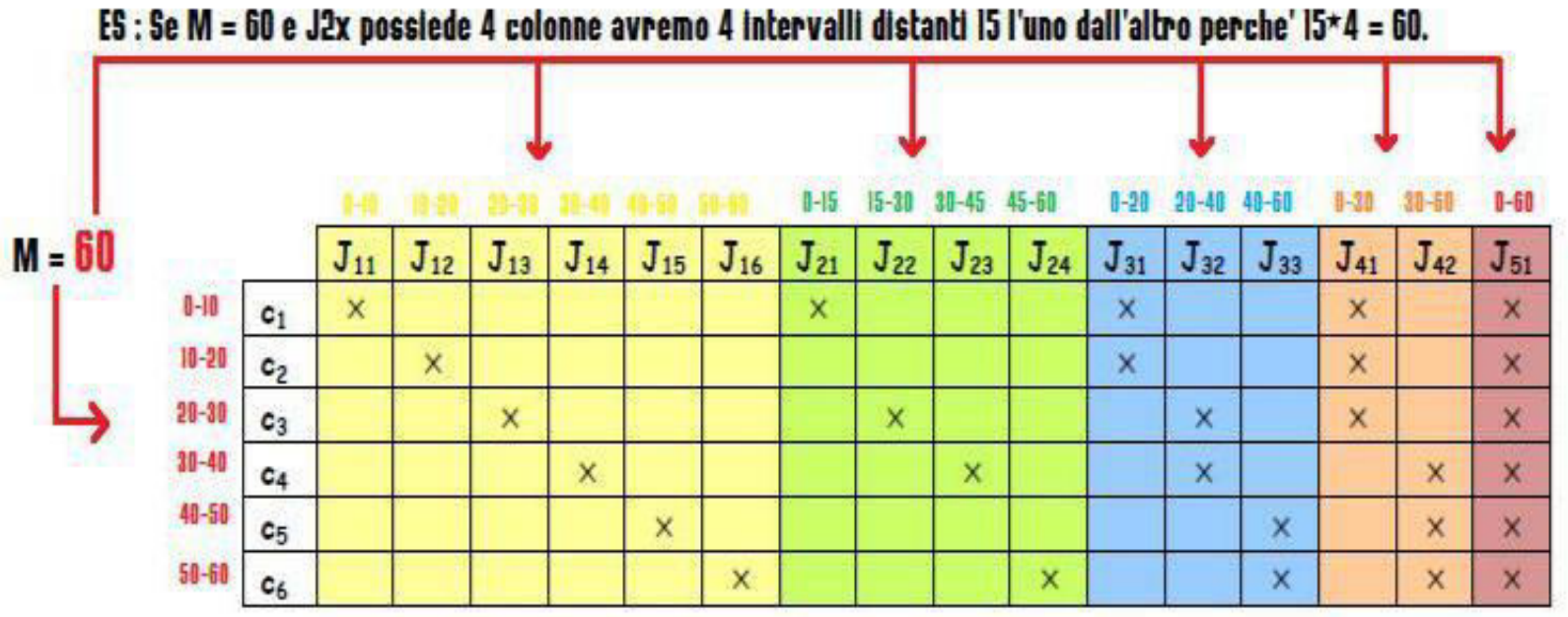
d) Identificazione del ciclo minore(o dei cicli minori) in cui ciascun Job può' essere eseguito:

Creare la seguente tabella con
Numero righe = ncm (calcolato al punto c)
Numero colonne = nj1+nj2+nj3+nj4+nj5.

Esempio:
Dato che nj1 = 6 Avremo J11,J12,J13,J14,J15,J16
Dato che nj2 = 4 Avremo J21,J22,J23,J24 etc etc…

Crocettare l'intersezione nella tabella solamente se l'insieme della RIGA considerata rappresentante i Ci è INTERAMENTE CONTENUTA nell'insieme della COLONNA. ES:

1)c1 ha come intervallo 0-10 mentre J11 ha come intervallo 0-10.
Essendo 0-10 interamente contenuto in 0-10 allora crocetto.
2)c3 ha come intervallo 20-30 mentre J21 ha come intervallo 0-15.
Essendo 20-30 NON interamente contenuto in 0-15 allora NON crocetto.
Crocetterò per esempio J22 in quanto 20-30 è interamente contenuto in 15-30.



e) Pianificazione dell'esecuzione di ciascun Job

La tabella appena fatta consente di produrre la tabella rappresentativa dell'esecuzione di ogni JOB.
Considerare che:
- Il numero di slot disponibili nella tabella (il numero di caselle bianche in cui posso piazzare un Job per ogni riga) è pari ad **m.**
- Ogni Jx,x deve occupare nella tabella tanti posti **quanti ne ha il suo rispettivo cx del testo del problema iniziale.**
Nel caso dell'esempio avremo che il posto occupato nella tabella da:

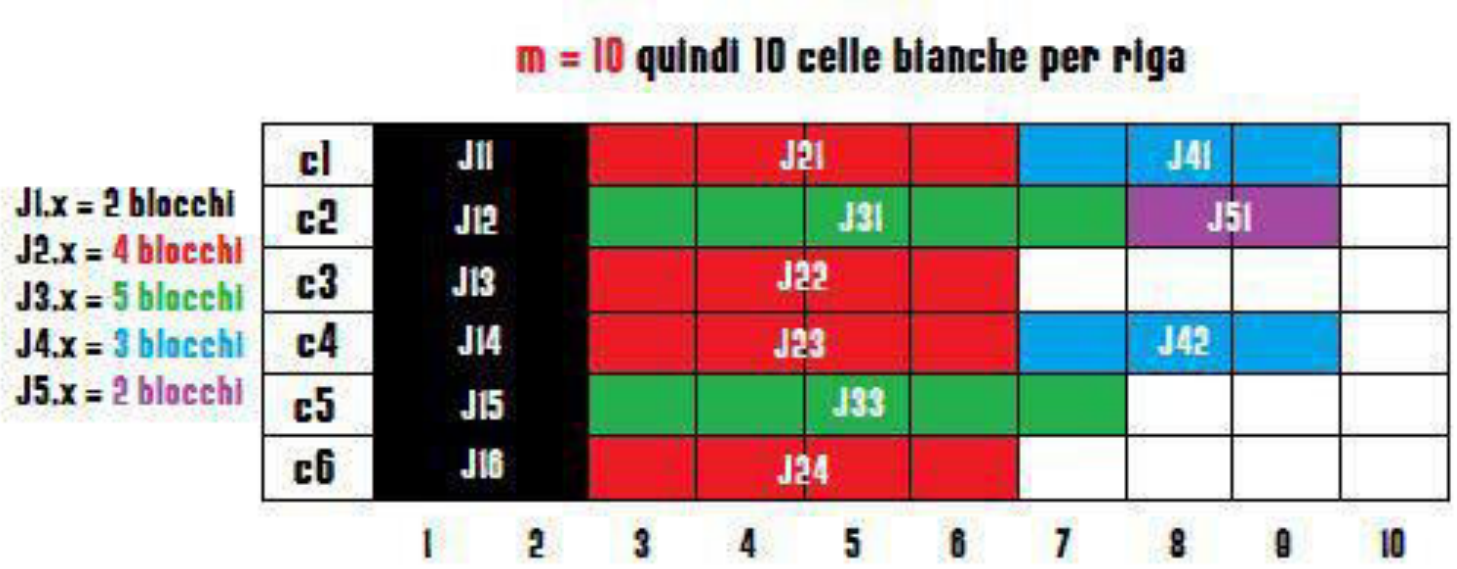
J1,x sarà di 2 blocchi (C1 = 2)
J2,x sarà di 4 blocchi (C2 = 4)
J3,x sarà di 5 blocchi (C3 = 5)
J4,x sarà di 3 blocchi (C4 = 3)
J5,x sarà di 2 blocchi (C5 = 2)

Per inserire un Job nella tabella attuale, occorre **controllare la tabella del punto d** e **verificare dove sono state inserite le crocette.**

Esempio: **J11** sarà inseribile nella tabella SOLO nella riga **C1** mentre J31 potrà essere inserito sia in C1 che in C2. Analogamente J41 potrà essere piazzato in C1,C2 oppure C3. Partire con l'inserimento dai **JOB più prioritari, ovvero i Job J1,X. Passare poi ai J2,X e così via.**
CONSIGLIO: Per evitare di dimenticarsi di posizionare in tabella uno qualsiasi dei Job,

segnarsi a lato la LISTA dei Job da schedulare nel seguente modo
J11,J12,J13,J14,J15,J16
J21,J22,J23,J24
J31,J32,J33,J34
J41,J42
J51

Ogni volta che si riesce a piazzare un Job CANCELLARLO dalla lista in modo da far rimanere solamente i Job non piazzati (se presenti) al termine dell'esercizio. Preferire il riempimento di una riga intera piuttosto che lasciare spazi bianchi a caso.



Nell'esempio era meglio mettere J41 in C2 in quanto si riempiva completamente lo spazio per quella riga!
Dopo aver definito la tabella ci sarà da rispondere se l'esito è stato positivo per **tutti i Job.**
In questo caso la risposta dipende dall'eventuale presenza di Job non inseriti (a causa della **mancanza di spazio**) nella tabella di sopra.
J32 per esempio e i suoi 5 spazi, sarebbe dovuto essere inserito in C3 oppure in C4 ma purtroppo lo spazio non era sufficiente.
In tal caso **si crocetta NO** e si indica **J32** come Job non inserito.

sì	no	quali job ?
	×	J_{32}

f) In caso di esito negativo, effettuare il rilassamento del minor numero possibile di vincoli

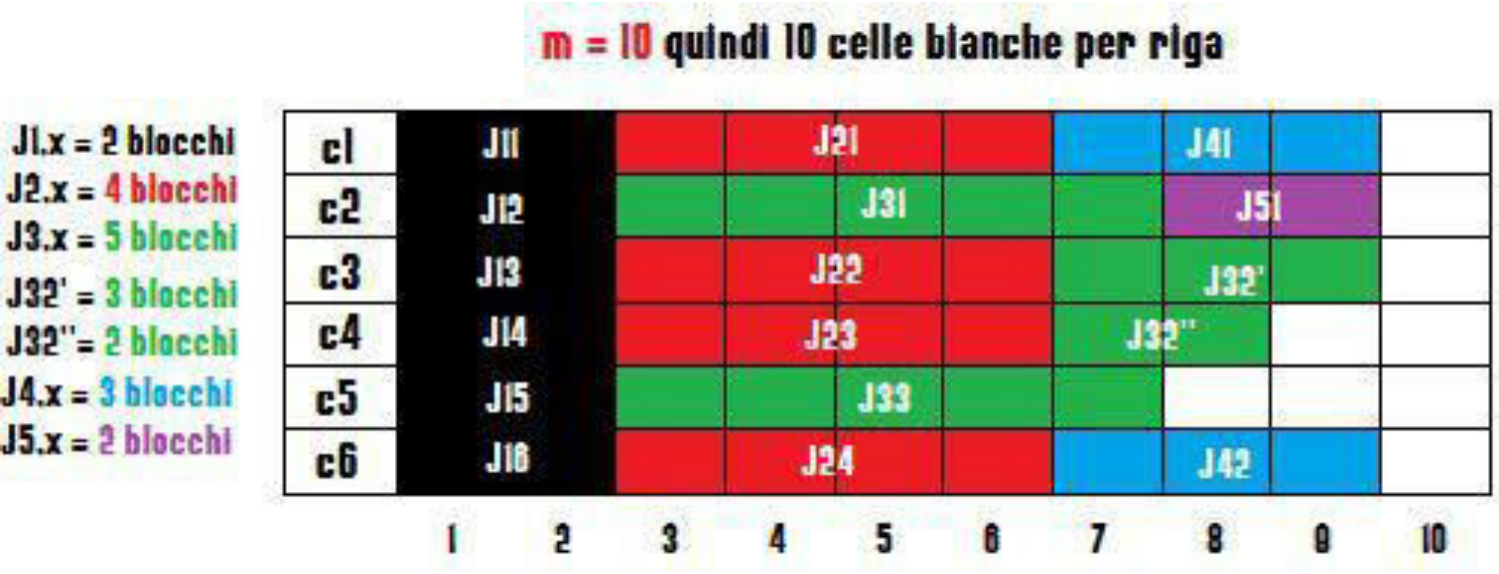
In questa sezione scrivere **Px (Ti,Ci)** del processo che non è stato schedulato e **spezzarlo come ci pare, in maniera da farlo entrare nella tabella.**

ATTENZIONE, TUTTO P3 VERRÀ SPEZZATO QUINDI AVREMO DUE NUOVI JOB DA SCHEDULARE ovvero J32' e J32''.

Puoi scegliere tu in che modo spezzare (qui si spezza un Ti = 5 in due Ti pari a 3 e il secondo pari a 2).
Nel caso dell'esempio avremo quindi **J32 che fa parte di P3**, quindi:

P3 (5,20) => P3' (3,20), P3''(2,30) con P3' < P3''

Rifare la tabella considerando l'esistenza di J32' e J32'' di occupazione blocchi pari a:
J32'--> 3 blocchi al posto dei precedenti 5
J32'' --> 2 blocchi al posto dei precedenti 5



NB: cercare di riempire le celle il più possibile. Se puoi riempire una riga, fallo! Per esempio qui J41 può tranquillamente andare a C2 così da riempire la riga, idem J51 a C4.

NB2: si può anche dividere in modo tale che 3 job da 4 diventino 4 job da 3

NB3: più job dello stesso processo che devono essere divisi, vanno suddivisi nella stessa maniera (ad esempio: se mi rimangono fuori J71 e J72 con Ci = 10, non posso suddividerli a J71' = 5, J71'' = 5 e J72' = 7 e J72'' = 3, ma o sono entrambi 5/5 o sono entrambi 7/3. Ricorda che è lo stesso processo quello che stai suddividendo)

Test di Han

	T_i [t.u.]	C_i [t.u.]
P_1	10	2
P_2	15	4
P_3	20	5
P_4	30	3
P_5	60	2

Occorre definire l'insieme accelerato dei processi. Partire dai dati forniti dal problema relativi ai Ti e ai Ci di partenza.
L'obiettivo è trovare un insieme di processi la cui **sommatoria degli Ui sia <=1.**
Per far passare il test di Han occorre infatti calcolare prima tutti gli Ui come -> Ui = Ci / Ti.
Il test di Han è positivo se

$$U_p = \sum_{i=0}^{n} \frac{C_i}{T_{i'}} \leq 1$$

FARE QUESTI CONTI SU UN FOGLIO DI BRUTTA, COPIARE SOLAMENTE L'ULTIMA TABELLA DOVE RISULTA CHE LA SOMMATORIA E' <= 1.
Prendere la tabella soprastante ed eseguire diverse **iterazioni** finchè la sommatoria degli Ui non risulta <=1.

Prendere ogni riga (dalla riga di P1 alla riga di P5) e tenerla **bloccata** al fine di calcolare i nuovi Ti **SOTTO** oppure di **SOPRA** che risulteranno (forse) modificati. NB: i Ci restano gli stessi per tutte le righe!

P_1	T_1	↑ T_1'
...	...	↑ ...
P_k	T_k	$T_k'=T_k$
...	...	↓ ...
P_N	T_N	↓ T_N'

↑ $T_i' = T_{i+1}' / \lceil T_{i+1}' / T_i \rceil$

↓ $T_i' = T_{i-1}' * \lfloor T_i / T_{i-1}' \rfloor$

Se **SALI** approssimi per **ECCESSO (^ parte BLU)**

Se **SCENDI** approssimi per **DIFETTO (v parte VERDE)**

Notare che nella formula ogni riga successiva è calcolata usando Ti-1' quindi il **risultato ottenuto nella riga precedente!**

Esempio con la tabella di sopra:
1° iterazione -> Tengo bloccata la riga con P1 (solo iterazioni per calcolare nuovi Ti verso il basso in quanto sopra P1 non ho nulla!)
Tengo bloccata la riga di P1
Calcolo il Ti' di P2 -> 10 * [15/10] = 10 * 1 = 10
Calcolo il Ti' di P3 -> 10* [20/10] = 10 * 2 = 20
Calcolo il Ti' di P4 -> 20* [30/20] = 20 * 1 = 20
Calcolo il Ti' di P5 -> 20* [60/20] = 20 * 3 = 60

S'	Ti'	Ci	Ui'
P1' = P1	10	2	0.2
P2'	10	4	0.4
P3'	20	5	0.25
P4'	20	3	0.15
P5'	60	2	0.033

NB: ricorda che ogni **Ui** per ogni riga = **Ci / Ti.**

$$U_p = \sum_{i=0}^{n} \frac{C_i}{T_{i'}} \leq 1$$

Calcolo di Up -> 2/10 + 4/10+ 5/20 + 3/20 + 2/60 = 1.03 > 1 NO -> Ci serve un'altra iterazione

Tengo bloccata la riga di P2. (Ti = 15)

Calcolo il Ti' di P1 -> 15 / [15/10] = 15 * 2 = 7.5 (*)
Calcolo il Ti' di P3 -> 15 * [20/15] = 15 * 1 = 15
Calcolo il Ti' di P4 -> 15 * [30/15] = 15 * 2 = 30
Calcolo il Ti' di P5 -> 30 * [60/30] = 30 * 2 = 60

(*)Se avessi bloccato la riga di P3 e avessi ottenuto Ti' =7.5 in tale riga, salendo verso l'alto avrei sempre riportato il valore 7.5 analogamente a come visto scendendo verso il basso. Essendo però in questo caso bloccata la Riga 2, sopra ho solamente una riga, quindi l'iterazione termina immediatamente usando i valori di P2.
Calcolo di Up -> 2/7.5 + 4/15 + 5/15 + 3/30 + 2/60 = 1 <= 1 **OK** -> **Ci possiamo fermare qui**

S'	Ti'	Ci	Ui'
P1'	7.5	2	0.267
P2' = P2	15	4	0.267
P3'	15	5	0.333
P4'	30	3	0.1
P5'	60	2	0.033

PASSATO, Quindi crocettare SI/SI.

Test di Kuo-Mok e corollario di Liu-Layland

Partire dalla tabella INIZIALE del testo della quale si sono calcolati i rispettivi Ui = Ci/Ti.

PROCESSO	T_i	C_i	$U_i = C_i/T_i$
P1	10	2	0.2
P2	15	4	0.27
P3	20	5	0.25
P4	30	3	0.1
P5	60	2	0.03

Effettuare dei raggruppamenti tra i processi, in una nuova tabella tale per cui la somma di tutti gli Ui appartenenti a quel gruppo è il più distante possibile dalla somma di tutte le altre Ui appartenenti ad altri gruppi.
NB: I processi che si decidono di unire nello stesso gruppo devono avere il più piccolo Ti del gruppo che è **MULTIPLO DI TUTTI GLI ALTRI Ti** appartenenti allo stesso gruppo.

Esempio: In questo esempio le possibili permutazioni tali per cui il Ti più piccolo (di ogni gruppo) risulta multiplo di tutti gli altri sono 3 (evitando quella in cui avremmo un gruppo singolo che è da evitare).

Relazione armonica: i processi sono di Ti multipli interi fra di loro (l'uno con l'altro, vicendevolmente). Per esempio {10, 20, 60} (10 è multiplo di 20 e 60, anche 20 è multiplo di 60), invece {10, 20, 30} non va bene, siccome 30 non è multiplo di 20.

Il processo P5 di Ti 60 è in relazione armonica con entrambi i gruppi {10, 20} e {15, 30} (che sono "obbligati"). Lo affidiamo al gruppo con maggiore fattore di utilizzazione Ui.
{P1, P3} hanno un Ui' (totale) di = 0.2+0.25 = 0.45
{P2, P4} hanno un Ui' (totale) di = 0.27+0.1 = 0.37

P5 può stare sia in {P1, P3}, sia in {P2, P4}, **Io metto nel gruppo con Ui' più alto**, quindi in {P1, P3}.
Risultato e gruppi in relazione armonica: **{P1, P3, P5}** e **{P2, P4}**.

La tabella così trovata avrà:

S'	T_i' [t.u.]	C_i' [t.u.]	U_i'
$P_1' \equiv \{P_1, P_3, P_5\}$	10	4.83	0.483
$P_2' \equiv \{P_2, P_4\}$	15	5.5	0.367

1) **Come Ui'** è la somma dei vari Ui dei processi appartenenti al gruppo

2) **Come Ti'** è il Ti più piccolo del processo in quel gruppo

3) **Come Ci'** si trova facendo la formula inversa ovvero Ui' * Ti' per ogni riga! (per la prima ad esempio Ci'/ Ti' = Ui' => 4.83/10 = 0.483

Si procede a scrivere la formula richiesta dal prof.
ATTENZIONE => E' possibile avere questo esercizio in 2 differenti varianti:

1) SOLO test di Kuo-Mok (se chiede solo lui, di solito fallisce)

Test Kuo-Mok:

$U_p \leq U_{RMPO}(N) = N(2^{1/N} - 1)$, Se:

- **Vero, indicare SI/SI (oppure procedere con Liu-Layland)**
- **Falso, indicare NO/NO**

NB: N=numero di gruppi (di processi). Nel nostro caso N = 2 in quanto abbiamo un gruppo con 3 processi e uno con 2, in totale 2 gruppi.
Valori noti:

- $U_{RMPO}(N = 2) = 0.828$
- $U_{RMPO}(N = 3) = 0.78$
- $U_{RMPO}(N = 4) = 0.757$
- $U_{RMPO}(N = 5) = 0.743$

2) Test di Kuo-Mok + corollario di Liu-LayLand (eseguire quanto visto al punto 1 e fare anche questo test sottostante)

Corollario del teorema di Liu-Layland:

$\prod_{i=1}^{N}(1 + U_i') \leq 2$ Se:

- **Vero, indicare SI/SI**
- **Falso, indicare NO/NO**

Nell'esempio:

- Kuo-Mok:$0.45 + 0.37 = 0.82 \leq 2(2^{1/2} - 1) = 0.828 \Rightarrow 0.82 \leq 0.828 \Rightarrow$ **OK**
- Liu-Layland: $[(1 + 0.45) * (1 + 0.37)] \leq 2 \Rightarrow 1.9865 \leq 2 \Rightarrow$ **OK**
- **Risposta: SI/SI**

Test di Burchard

$$X_j = \log_2 T_j - \lfloor \log_2 T_j \rfloor, \ \forall j$$

$$\zeta = \max_{1 \le j \le N} X_j - \min_{1 \le j \le N} X_j$$

$$U_{RMPO}(N, \zeta) = \begin{cases} (N-1)(2^{\frac{\zeta}{(N-1)}} - 1) + 2^{1-\zeta} - 1 & \zeta < 1 - \frac{1}{N} \\ N(2^{\frac{1}{N}} - 1) & \zeta \ge 1 - \frac{1}{N} \end{cases}$$

Test: $U_P \le U_{RMPO}$ **Se:**

- **Vero, indicare SI/SI**
- **Falso, indicare NO/NO**

Esempio con la seguente configurazione di processi (ricorda arrotonda per DIFETTO)
NB: N = numero dei processi quindi = 4 in questo esempio.

	T_i	C_i	U_i
P1	4	1	0.25
P2	6	1	0.17
P3	8	2	0.25
P4	12	2	0.17

$X_1 = log_2(4) - \lfloor log_2(4) \rfloor = 0$ $X_2 = log_2(6) - \lfloor log_2(6) \rfloor = 0.585$

$X_3 = log_2(8) - \lfloor log_2(8) \rfloor = 0$ $X_4 = log_2(12) - \lfloor log_2(12) \rfloor = 0.585$

$= 0.585 - 0 = 0.585$

$U_P = 0.84 \le U_{RMPO}(N,) = 0.768 \Rightarrow$ **NO**

Risposta: NO/NO

Algoritmo di Audsley

Quando il numero di processi è elevato (come nell'esempio preso in considerazione) è possibile che l'algoritmo di Audsley non venga richiesto.

Per questo esempio prenderemo in considerazione i seguenti processi.

	T_i [t.u.]	C_i [t.u.]
P_1	4	1
P_2	6	1
P_3	8	2
P_4	12	2

Scorrere la tabella VERTICALMENTE da sinistra verso destra, fermarsi nella colonna relativa ad ogni processo se, dopo aver effettuato il calcolo, il numero appena trovato risulta uguale al numero **sopra**. In caso contrario continuare ad applicare la formula per i numeri sottostanti.

1) La Riga **R0 = Ci** quindi basta semplicemente copiare i Ci del testo del problema.

2) Ogni altra riga è calcolata nel seguente modo:

Ri= Ri PRECEDENTE (rimane costante nella sommatoria)

Cj= Ci dei processi di indice MINORE

Tj= Ti dei processi di indice MINORE

Mi fermo appena ne ho 2 uguali per ogni colonna

$$I_i(R_j) = \sum_{j|p_j>p_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$R_i^0 = C_i$$

$$R_i^n = C_i + I_i(R_i^{n-1}), \quad n = 1, 2, \ldots$$

Che in italiano significa semplicemente: (NB approssimazione **PER ECCESSO**)

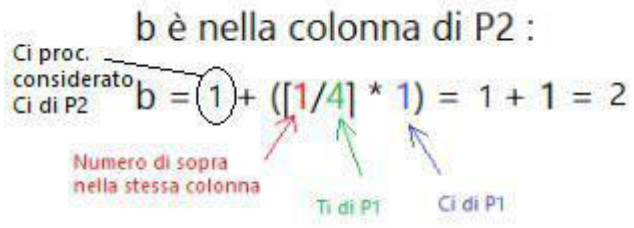
R^0_i +
($\lceil (numeroDiSopra/T_{iProcessoConsiderato-1}) \rceil$ * $C_{iProcessoConsiderato-1}$) +
($\lceil (numeroDiSopra/T_{iProcessoConsiderato-2}) \rceil$ * $C_{iProcessoConsiderato-2}$) + …

Si tratta di una **sommatoria**, quindi nel caso di P1 il valore subito sotto sarà pari al valore subito sopra (in quanto non esistono processi più prioritari).
Per P2 saranno da sommare i valori relativi a P1, per P3 avremo sia P2 che P1 mentre per P4 avremo tutti i processi nella sommatoria.

Vediamo con un esempio:

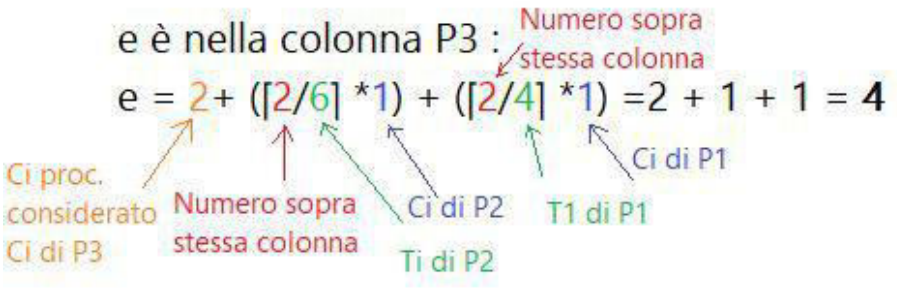
	P1	P2	P3	P4
$R_i^0 = C_i$	1	1	2	2
$R_i^1 = C_i + I_i(R_i^0)$	a = 1	b = 2	e = 4	g = 6
$R_i^2 = C_i + I_i(R_i^1)$		c = 2	f = 4	h = 7
$R_i^3 = C_i + I_i(R_i^2)$				i = 8
$R_i^4 = C_i + I_i(R_i^3)$				j = 8

a = basta guardare il numero di sopra. L'algoritmo termina sempre per il caso a, in quanto ho 2 numeri uguali nella colonna immediatamente



Essendo 2, risultato appena trovato, diverso da 1 (numero sopra) devo continuare e fare una successiva iterazione. Attenzione, il numero di sopra della colonna ora diventa pari a 2!

c = 1+(2/4 * 1) = 1 + 1 = 2 **STOP**



Qui notiamo che i processi più prioritari sono 2.
Quindi, trovandomi nella colonna di P3 dovrò considerare un pezzo relativo a P2 ed uno relativo a P1.

f = 2+ ($\lceil 4/6 \rceil$ * 1) + ($\lceil 4/4 \rceil$* 1) = 2 + 1 + 1 = **4** **STOP**

g = 2+ ($\lceil 2/8 \rceil$ * 2) + ($\lceil 2/6 \rceil$ * 1) + ($\lceil 2/4 \rceil$ * 1) = 2 + 2 + 1 + 1 = 6
h = 2+ ($\lceil 6/8 \rceil$ * 2) + ($\lceil 6/6 \rceil$ * 1) + ($\lceil 6/4 \rceil$ * 1) = 2 + 2 + 1 + 2 = 7
g = 2+ ($\lceil 7/8 \rceil$ * 2) + ($\lceil 7/6 \rceil$ * 1) + ($\lceil 7/4 \rceil$ * 1) = 2 + 2 + 2 + 2 = **8**
h = 2+ ($\lceil 8/8 \rceil$ * 2) + ($\lceil 8/6 \rceil$ * 1) + ($\lceil 8/4 \rceil$ * 1) = 2 + 2 + 2 + 2 = **8** **STOP**

Consiglio: Tenersi da parte la tabella con Ti e Ci. Alla fine per i calcoli successivi della stessa colonna basta modificare il dato relativo al **numero Sopra**. Ti e Ci restano uguali!

La RISPOSTA può essere SÌ/SÌ oppure NO/SI in base alla situazione ottenuta dopo la tabella: se mi fermo in tutte le colonne SI/SI.

Non strict LST

In pratica: dare priorità di esecuzione ai processi con slack minore. Lo slack mi dice quanto al massimo posso differire l'esecuzione.

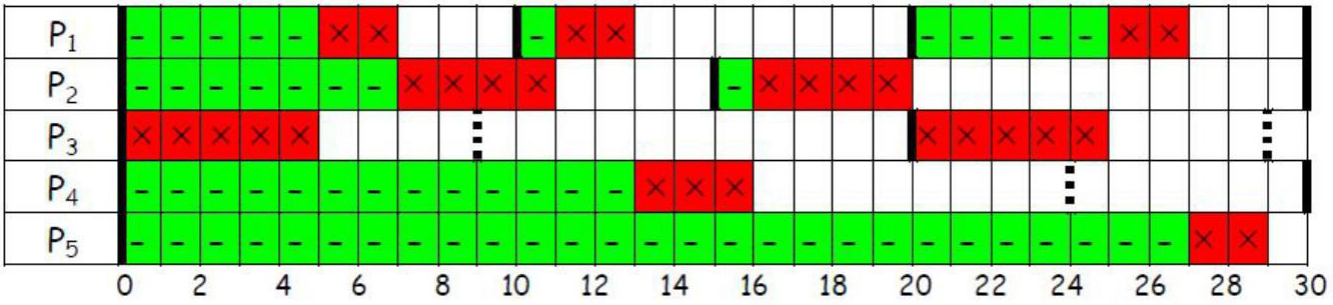
Il calcolo dello slack della tabella (e il conseguente valore t= per quella colonna) va rifatto ogni volta che si incontra un periodo (barra nera verticale).

Per questo algoritmo occorrono le Deadline fornite dal testo (se non specificata, la deadline è pari al Ti del processo), nell'esempio considerato abbiamo:

	T_i	C_i	D_i
P1	10	2	10
P2	15	4	15
P3	20	5	9
P4	30	3	24
P5	60	2	40

*Slack = * caselle prima della prossima deadline * − * caselle rosse ancora da eseguire **
Oppure: *Prossima deadline − t + < quante volte ho già eseguito in quel periodo > − C_i*

Il calcolo di una nuova colonna dello slack va fatta ogni volta che si incontra un nuovo periodo (di qualsiasi processo). In questo caso i primi periodi incontrati sono per t=0, t=10, t=15 e infine a t=20 (il prof ci dà lo spazio necessario per fare il giusto numero di istanti che gli interessano).
Per quanto riguarda la costruzione del grafico bisogna dare la priorità nell'esecuzione dei processi che possiedono lo SLACK più vicino.



slack	t=0	t=10	t=15	t=20
P_1	8	8	-	8
P_2	11	4	11	-
P_3	4	-	-	4
P_4	21	11	8	-
P_5	38	28	23	18

	T_i	C_i	D_i
P1	4	1	4
P2	8	3	8
P3	20	4	15
P4	40	2	10

Di - Ci

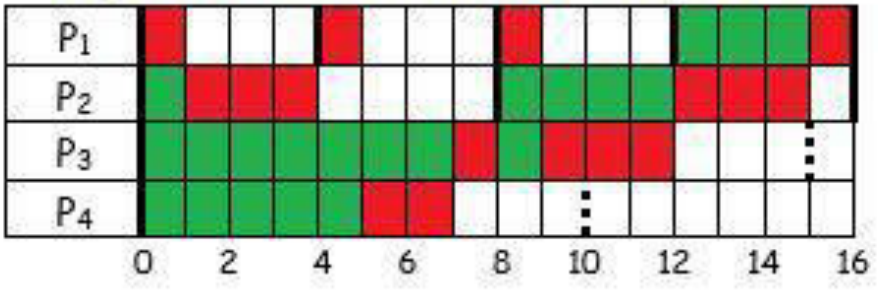
slack	t=0	t= 4	t= 8	t= 12
P_1	3	3	3	3
P_2	5	-	5	1
P_3	11	7	4	-
P_4	8	4	-	-

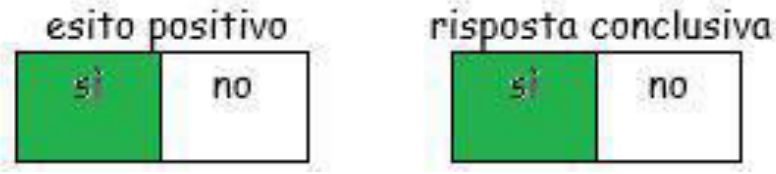
A t = 4 ho la prossima deadline tra 4 caselle bianche e devo ancora eseguire 1 in quel periodo 4-1 = 3

Se ho già eseguito e completato in quel periodo considerato, metti -

A t = 4 ho la prossima deadline tra 6 caselle e devo ancora eseguire 2 in quel periodo 6-2 = 4

Esegue SEMPRE chi ha lo slack calcolato più corto!





Da notare che per calcolare il t=0 la formula è semplicemente Di-Ci per il processo considerato.

La costruzione del grafico si effettua dando la priorità al processo con SLACK calcolato minore. Lui esegue, gli altri aspettano.

Il test passa (SI/SI) se TUTTI i processi sono riusciti ad eseguire prima della rispettiva deadline, in caso negativo crocettare NO/NO.
Attenzione alle deadline (per esempio quella di P4 o di P3) e a non confonderle con il periodo: un processo deve eseguire ENTRO la deadline, NON deve ri-eseguire superata la deadline!

Ricordarsi, se non presenti, di segnare le deadline sul grafico con dei tratteggi!

DMPO

Fattore di utilizzazione efficace del processore (Lehoczky)

Prendere la tabella originale con l'aggiunta dei Di e δi. Alcune Di sono indicate nel testo, le altre vengono messe pari a Ti. δi = Di / Ti

	Ti	Ci	Di	δi
P1	10	2	10	1
P2	12	1	12	1
P3	15	3	9	0.6
P4	20	6	14	0.7

Ordinare i processi in ordine crescente di Di (cioè in ordine di priorità) in una nuova tabella

	Ti	Ci	Di	δi
P3	15	3	9	0.6
P1	10	2	10	1
P2	12	1	12	1
P4	20	6	14	0.7

Definire una ulteriore nuova tabella con 4 colonne aggiuntive: H_n , H_1 , f, U(N, δ)

- *Hn* è il sottoinsieme di processi sopra al processo considerato Pj con *Ti < Dj*
- *H*1 è il sottoinsieme di processi sopra al processo considerato Pj con *Ti ≥ Dj*

$$f_j = \left(\sum_{i \in H_n} \frac{C_i}{T_i}\right) + \frac{1}{T_j}\left(C_j + \sum^{k \in H_i} C_k\right)$$

$$\sum_{j=1}^{N} \frac{C_j}{T_j} \leq U(N, \delta) = \begin{cases} N\left((2\delta)^{1/N} - 1\right) + 1 - \delta & 0.5 \leq \delta \leq 1 \\ \delta & 0 \leq \delta \leq 0.5 \end{cases}$$

Con

$$N = |H_n| + 1$$
$$\delta = \delta_i$$

ovvero la cardinalità di H_n (il numero di elementi) + 1.

Il test risulta passato se

$$f_i \leq U_i(N, \delta) \; \forall i \in P$$

NB: H_n , H_1 sono sempre vuoti per il primo processo considerato!

	H_n	H_n	f	U(N, δ)	f ≤ U(N, δ)
P3	{}	{}	0.2	0.6	sì
P1	{}	{P3}	0.5	1	sì
P2	{P1}	{P3}	0.53	0.828	sì
P4	{P1, P2}	{P3}	0.73	0.656	no

Per costruire gli insiemi ci si chiede se la Deadline (D) del processo considerato in esame è **maggiore** oppure **minore** di ogni Ti dei processi che si trovano sopra ad esso.

1) Se **maggiore o uguale** inserisco il processo più prioritario (quello sopra) in H_n

2) Se **minore** inserisco il processo più prioritario (quello sopra) in H_1

P3 => Non esistono processi più prioritari, gli insiemi H_n e H_1 sono entrambi vuoti.
La f vale, sempre e solo per la prima riga, Ci/Ti.
P1 => Confronto Di di P1(10) con Ti(15) di P3. Risulta 10 < 15 quindi P3 va in H_1
P2 => Confronto Di di P2(12) con Ti(15) di P3. Risulta 12 < 15 quindi P3 va in H_1
Confronto Di di P2(12) con Ti(10) di P1. Risulta 12 ≥ 10 quindi P1 va in H_n
P4 => Confronto Di di P4(14) con Ti(12) di P2. Risulta 14 ≥ 12 quindi P2 va in H_n

Se siamo nella prima riga la f è uguale alla U_i del processo della prima riga

$f(P3) = U_3 = Ci/Ti = 0.2$

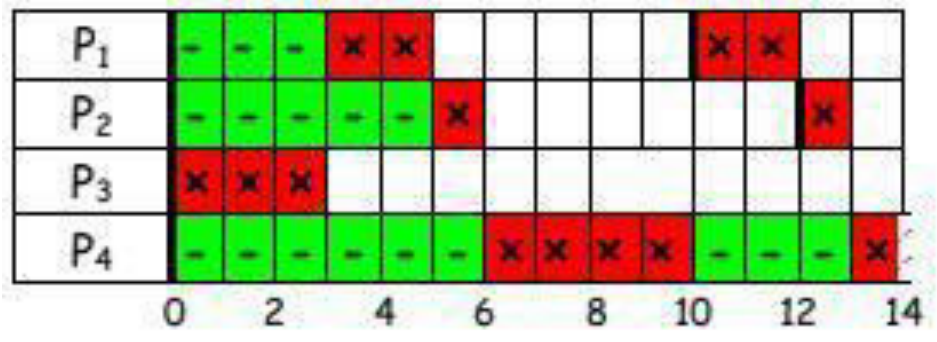
Altrimenti la formula va applicata per intero, considerando che T_j e C_j sono quelli della riga corrente
f(P1) = 1/10 * (2 + 3) = 0.5
f(P2) = 0.2 + 1/12 * (1 + 3) = 0.533
f(P4 0.2 + 0.083 + 1/20 * (6 + 3) = 0.733

Nel calcolo della U va considerato il δ_i corrente e come N il numero degli elementi in H_n +1
P3 =>U(1,0.6)= (2δ − 1) +1 − δ = δ = 0.6
P1 => U(1, 1) = 1
P2 => U(2, 1) = 2 * ((2 * 1)$^{1/2}$ − 1) + 1 − 1 = 0.828
P4 => U(3, 0.7) = 3 * ((2 * 0.7)$^{1/3}$ − 1) + 1 − 0.7 = 0.656

Se la colonna di f ≤ U(N, δ) contiene tutti sì la risposta è **SI/SI**, altrimenti è **NO/NO**

Diagramma temporale

	T_i	C_i	D_i	δ_i
P1	10	2	10	1
P2	12	1	12	1
P3	15	3	9	0.6
P4	20	6	14	0.7



Da notare come P4 dovrebbe eseguire 6 volte e ha come deadline 14.
Nel diagramma abbiamo eseguito solo 5 volte, quindi c'è una missed deadline

Se tutte le deadline sono state rispettate scrivere **SI/SI**, altrimenti **NO/SI**

EDF

Test basato sulla densità di utilizzazione del processore

$$\Delta = \sum_{i=1}^{N} \frac{C_i}{D_i} \leq 1$$

	T_i	C_i	D_i	C_i / D_i
P1	10	2	10	0.2
P2	12	1	12	0.083
P3	15	3	9	0.33
P4	20	6	14	0.428

Δ = 0.2 + 0.083 + 0.33 + 0.428 = 1.041 > 1 ⇒ NO
Se $\Delta \leq 1$ la risposta **SI/SI**,altrimenti **NO/NO**

Approccio processor demand

	T_i	C_i	D_i	C_i / D_i	U_i
P1	10	2	10	0.2	0.2
P2	12	1	12	0.083	0.083
P3	15	3	9	0.33	0.2
P4	20	6	14	0.428	0.3

Si comincia calcolando t*: dove:

$$t^* = \frac{\sum_{i=1}^{N}(1 - \frac{D_i}{T_i})C_i}{1 - U_p}$$

$$Up = \sum_{i=1}^{N} \frac{C_i}{T_i}$$

Up = 0.2 + 0.083 + 0.2 + 0.3 = 0.783

$$t^* = \frac{(1-1)*2+(1-1)*1+(1-0.6)*3+(1-0.7)*6}{1-0.783} = 13.82$$

Proseguire calcolando i BI:

$$BI^n = \sum_{i=1}^{N} \left\lceil \frac{BI^{n-1}}{T_i} \right\rceil C_i \qquad \text{con} \qquad BI^0 = \sum_{i=1}^{N} C_i$$

Ci si ferma quando si trovano due BI^n uguali, tale valore è detto BI.

$BI^0 = \sum_{i=1}^{N} C_i = 2 + 1 + 3 + 6 = 12 \rightarrow$
$BI^1 = \lceil 12/10 \rceil * 2 + \lceil 12/12 \rceil * 1 + \lceil 12/12 \rceil * 3 + \lceil 12/20 \rceil * 6 = 14 \rightarrow$
$BI^2 = \lceil 14/10 \rceil * 2 + \lceil 14/12 \rceil * 1 + \lceil 14/15 \rceil * 3 + \lceil 14/20 \rceil * 6 = 15 \rightarrow$
$BI^3 = \lceil 15/10 \rceil * 2 + \lceil 15/12 \rceil * 1 + \lceil 15/15 \rceil * 3 + \lceil 15/20 \rceil * 6 = 15$

n	BI^n
0	12
1	14
2	15
3	15

Si sceglie un valore $m = min(t^*, BI)$.
$m = min(13.82, 15) = 13.82$

Si calcola l'insieme $D \cap D^*$ che rappresenterà tutti i valori nella tabella alla colonna t .
Formalmente si definisce

$$D \cap D^* = \{d_{ik} \mid d_{ik} = (k - 1)\, T_i + D_i;\ d_{ik} < m,\ 1 \leq i \leq N \wedge k \geq 1;\ i, k \in \mathbb{N}\}$$

il che significa selezionare per ogni processo tutti quei valori strettamente minori di m ottenuti sommando la deadline del processo D_i a tante volte T_i finché ancora verificata tale condizione. In tale formula la k rappresenta semplicemente il valore su cui si cicla finché la condizione non è più verificata, mentre la i il processo preso in considerazione.

E.g. se si ha Ti = 8 e Di = 7 e t* = 26 , l'insieme D ∩ D* = {7, 15, 23} in quanto
Di = 7, Ti + Di = 7 + 8 = 15, Di + Ti + Ti = 23

Nell'esempio: $D \cap D^* = \{9, 10, 12\}$ (in ordine crescente)

A questo punto si può procedere con il riempimento della tabella seguendo la formula:

$$C_p(0,\ t) = \sum_{i=1}^{N} C_i(0,\ t) = \sum_{i=1}^{N} \left(\lfloor \frac{t-D_i}{T_i} \rfloor + 1 \right) C_i$$

La tabella è riempita scrivendo in ogni riga il risultato di

$$C_i(0,\ t) = \left(\lfloor \frac{t-D_i}{T_i} \rfloor + 1 \right) C_i$$

con t l'elemento corrente dell'insieme $D \cap D^*$

NB: Si ricorda che l'arrotondamento per difetto di un numero negativo, prende il numero negativo intero più piccolo (es. $\lfloor -0.1 \rfloor = -1$)

Nell'esempio:

$$C_1(0,\ 9) = (\lfloor \frac{9-10}{10} \rfloor + 1) * 2 = (\lfloor -0.1 \rfloor + 1) * 2 = (-1 + 1) * 2 = 0$$
$$C_2(0,\ 9) = (\lfloor \frac{9-12}{12} \rfloor + 1) * 1 = (\lfloor -0.25 \rfloor + 1) * 1 = (-1 + 1) * 1 = 0$$
$$C_3(0,\ 9) = (\lfloor \frac{9-9}{15} \rfloor + 1) * 3 = (\lfloor 0 \rfloor + 1) * 3 = 1 * 3 = 3$$
$$C_4(0,\ 9) = (\lfloor \frac{9-14}{20} \rfloor + 1) * 6 = (\lfloor -0.25 \rfloor + 1) * 6 = (-1 + 1) * 6 = 0$$
$$C_p(0,\ 9) = C_1(0,\ 9) + C_2(0,\ 9) + C_3(0,\ 9) + C_4(0,\ 9) = 0 + 0 + 3 + 0 = 3$$

t	C_1 (0, t)	C_2 (0, t)	C_3 (0, t)	C_4 (0, t)	C_p (0, t)	≤t
9	0	0	3	0	3	Sì
10	2	0	3	0	5	Sì
12	2	1	3	0	6	Sì

Se la colonna di ≤ t contiene tutti sì la risposta è **SI/SI**, altrimenti è **NO/NO**

PROBLEMA 2

In questo problema occorre gestire delle richieste asincrone (Ra) che vengono eseguite in diversi modi. I grafici vanno compilati da sinistra verso destra, un'istante alla volta, contraollando chi ha priorità in quell'istante

NB: SEGNARE IN TUTTI I GRAFICI ANCHE L'ISTANTE IN CUI UNA RICHIESTA ASINCRONA VIENE COMPLETATA (SERVE PER IL SERVER CUS!)

RMPO

Servizio in background

	a_i [t.u.]	C_i [t.u.]
R_{a1}	5	3
R_{a2}	15	4
R_{a3}	25	2
R_{a4}	35	1

	T_i [t.u.]	C_i [t.u.]
P_1	8	2
P_2	10	1
P_3	12	2
P_4	15	2

	P_i	R_a	S
	Idle	None	Idle
-	Ready	-	Ready
...	-	Pending	Waiting
X	Running	Being Served	Running

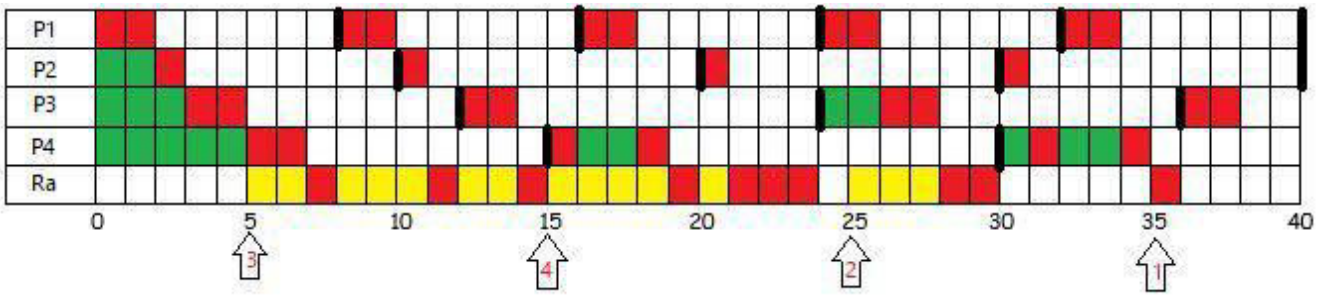
Le richieste asincrone sono trattate come processi di priorità minima.
In questo tipo di servizio NON è presente un vero e proprio server:

1. Segnare prima in rosso (per ogni periodo) l'esecuzione del processo P1 in quanto di priorità maggiore.
2. Fare analogamente con tutti gli altri processi all'istante successivo disponibile.
3. Le richieste asincrone sono da gestire come processi di priorità **minima**.

NB: Ordinare i processi dal Ti più basso al Ti più alto (se non già fatto dal testo) in quanto rappresenta l'ordine di esecuzione dei processi stessi.

SIMBOLOGIA:

- **ROSSO** = Per rappresentare l'esecuzione sia di processi che di richieste asincrone.
- **VERDE** = Se il processo vorrebbe eseguire ma non può eseguire perché altri lo stanno già facendo. In sintesi colora tutti gli spazi PRIMA di tutti i rossi in un periodo.
- **GIALLO** = SOLO per le richieste asincrone, rappresenta la volontà di una Ra di eseguire ma l'impossibilità di farlo (attenzione a NON segnarle se non sono arrivate / non ci sono!)
- **BIANCO**= Quando ho terminato le mie esecuzioni e sono in attesa del prossimo periodo (per un processo) o di una successiva richiesta (per una Ra)



Polling server

In questo caso c'è un server che ha come compito l'esecuzione delle richieste asincrone. Il server ha una sua **capacità Cs e un Periodo Ts**, che indica se il server può o meno eseguire.

- La capacità viene consumata durante il servizio di richieste asincrone
- La capacità viene completamente scaricata (**azzerata**), anche se è positiva, quando non sono presenti richieste asincrone.
- ATTENZIONE: La priorità del server **DIPENDE** dal suo T_S , quindi **si consiglia di riscrivere la Tabella dei processi, inserendo il Server** e il suo Ts alla priorità corretta.
- La capacità del server TORNA A LIVELLO MASSIMO ogni suo periodo.

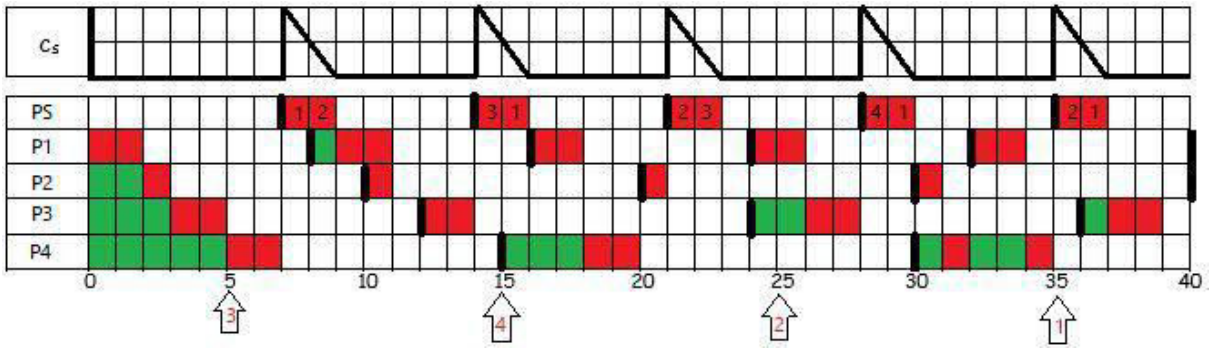
Segnati quante unità di esecuzione hai eseguito di ogni richiesta per non confonderti

	T_i [t.u.]	C_i [t.u.]
P_1	8	2
P_2	10	1
P_3	12	2
P_4	15	2

	a_i [t.u.]	C_i [t.u.]
R_{a1}	5	3
R_{a2}	15	4
R_{a3}	25	2
R_{a4}	35	1

$T_S = 7 \quad C_S = 2$

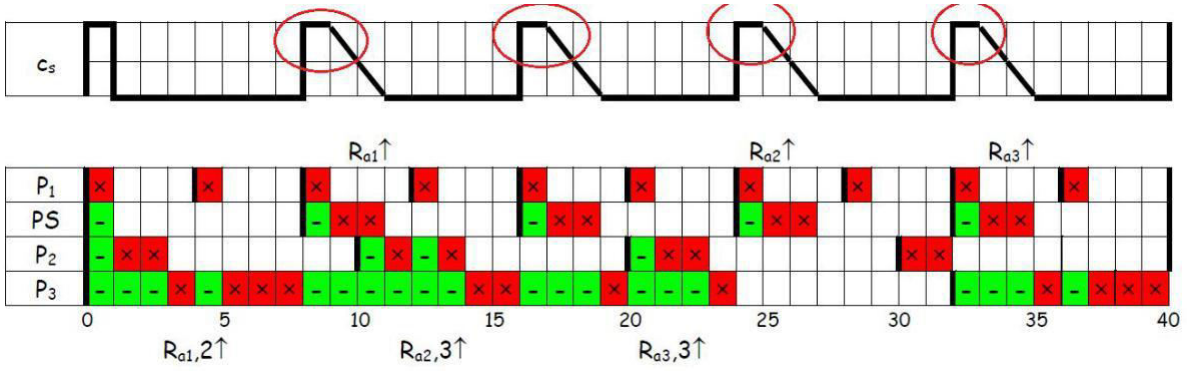
NB: La capacità del server nel grafico è alta quanto C_S e parte sempre dal valore massimo
Il server non sarà mai in stato Pending (**GIALLO**), siccome il server consuma la capacità



CASO PARTICOLARE:
In questo caso particolare il server NON possiede assoluta priorità in quanto il suo Ts = 8 è maggiore di quello di P1 che vale 4.

Essendo P1 quindi più prioritario, occorre **MANTENERE LA CAPACITÀ del server finchè esso non può eseguire**.

Se avessi avuto 2 processi più prioritari del server, avrei mantenuto la capacità anche per più unità temporali.



Deferrable server

Come il polling, ma la capacità **si conserva** invece di scaricarsi in assenza di richieste.
Ricordarsi di ricaricare la capacità del server ogni suo periodo.

	T_i [t.u.]	C_i [t.u.]
P_1	10	3
P_2	15	3
P_3	35	7

	a_i [t.u.]	C_i [t.u.]
R_{a1}	4	1
R_{a2}	9	2
R_{a3}	19	3
R_{a4}	29	2
R_{a5}	36	1

$T_s = 8$ $C_s = 2$

Il Deferrable Server va segnato di **GIALLO** quando la sua capacità in qualsiasi punto del grafico risulta MAGGIORE O UGUALE A 1. Esiste un caso particolare in cui ciò non avviene e invece si colora di **VERDE** ovvero se il server non ha priorità massima.

Priority Exchange Server

In questo tipo di esercizio, la priorità all'esecuzione va data ai processi con il Ti minore. Questo implica che essi debbano essere ordinati in ordine di priorità (server incluso, il cui Ts è fornito dal testo). Il processo più in alto ha la priorità maggiore.

	T_i [t.u.]	C_i [t.u.]
P_1	10	3
P_2	15	3
P_3	35	7

	a_i [t.u.]	C_i [t.u.]
R_{a1}	4	1
R_{a2}	9	2
R_{a3}	19	3
R_{a4}	29	2
R_{a5}	36	1

NB: LA CAPACITÀ' INDICA LA POSSIBILITA DEL SERVER DI ESEGUIRE, NON PREGIUDICA L'ESECUZIONE DI PROCESSI.
I processi eseguono normalmente in base alla loro priorità, ma se sto eseguendo e un processo di priorità superiore ha della capacità, gliela rubo.
La capacità totale nel sistema diminuisce (partendo dal più alto) solo se il server esegue o se tutto il sistema è idle, in caso contrario viene conservata tra i vari processi.
Il server può eseguire solo se c'è della capacità nel sistema. In tal caso inizia a usare quella del processo più prioritario.

Nel Priority Exchange Server il server ha una sua capacità che inizia al massimo e che viene ricaricata ogni Ts, essa può essere **ceduta** ai processi di priorità inferiore (quelli sotto). Quindi la capacità va sempre verso il basso.

La cessione della capacità può essere:

-Dal **Server** a **processi di priorità inferiore al server** -> Avviene quando il server **NON ha richieste aperiodiche da gestire** e i processi di sotto (almeno uno) eseguono.
-Da processi superiori a processi inferiori che eseguono (se ho più gente sopra si prende da quello più in alto, anche NON adiacenti)

Se tra P1, P2 e P3 esegue P2 e P1 ha della capacità, allora P2 ruba capacità a un processo che gli sta sopra aumentando la propria capacità da 0 a 1.

Analogamente, all'istante dopo, eseguendo P3 egli può (e lo fa) rubare capacità ad un processo che gli sta sopra aumentando la propria capacità da 1 a 2.

P1 C' = 1	C = 0	C = 0
P2 C' = 0 <	C = 1	C = 0
P3 C' = 1	C = 1 <	C = 2

$\forall p \in P_{run}\ C_p = C'_p + 1$ *se* $\exists\, p' \in P : p' \gg p \wedge C_{p'} > 0;\ C_{p'} = C'_{p'} - 1$ *altrimenti* $C_p = C'_p$

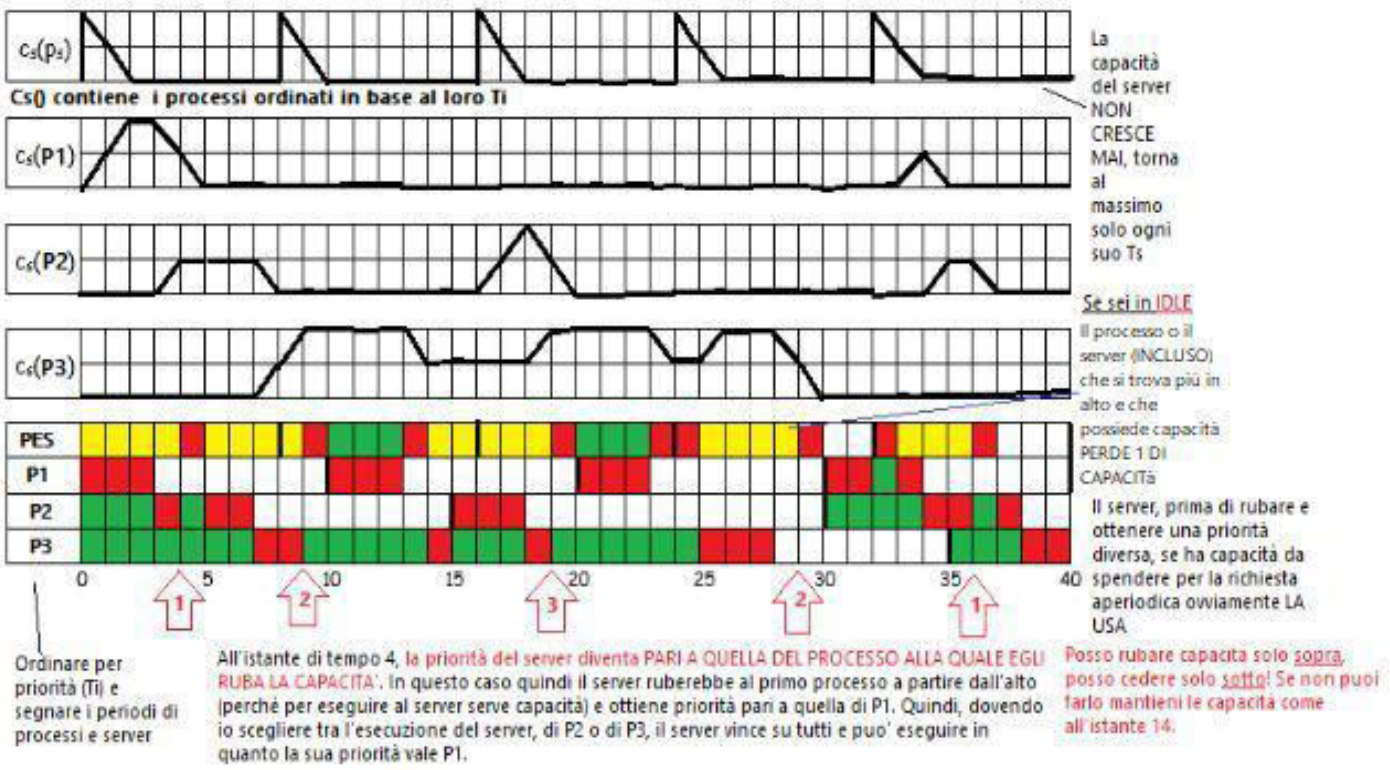
se P ES in esecuzione $\wedge\ C_{pes} = 0 \wedge \exists\, p : C_p > 0 \Rightarrow C_p = C'_p -$

1 *se P ES in esecuzione* $\wedge\ C'_{pes} > 0 \Rightarrow C_{pes} = C'_{pes} - 1$

se $\exists p \in P_{run} \exists !\ p \in P : C_p > 0\ C_p = C'_p - 1$

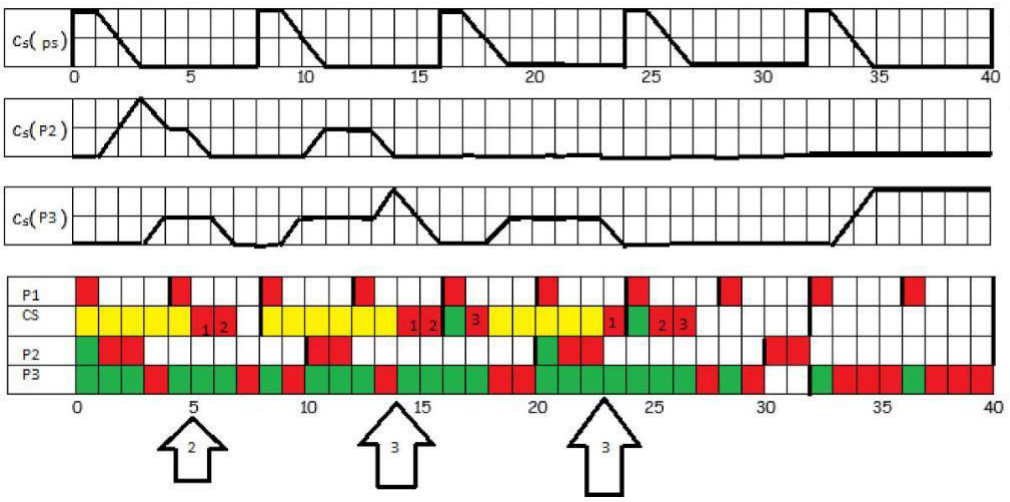
- Un processo che esegue mantiene (o ruba) la propria capacità
- La capacità si mantiene anche nel caso in cui esistano processi di priorità SUPERIORE

Il server è sempre di colore **GIALLO** quando c'è capacità ovunque > 0.
Se la capacità è nulla, il colore è **BIANCO**.
Invece, se il server ha capacità e potrebbe eseguire una richiesta aperiodica, ma è stato bloccato da un processo di priorità superiore, il colore è **VERDE**.



CASO PARTICOLARE: IL SERVER NON E' IL PROCESSO PIÙ PRIORITARIO, IN TALI CASI, SE QUALCUNO DI PIÙ PRIORITARIO ESEGUE,LA SUA CAPACITÀ VIENE MANTENUTA.

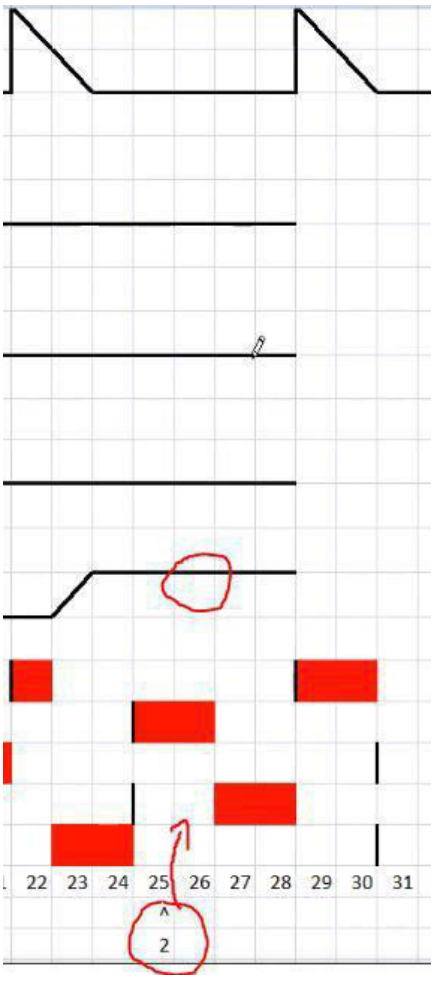
INOLTRE NON OCCORRE DISEGNARE IL GRAFICO DI cs(P1) in quanto la capacità può andare solo verso il basso.



	T_i [t.u.]	C_i [t.u.]
P_1	4	1
P_2	10	2
P_3	32	8

Cs = 2
Ts = 8

	a_i [t.u.]	C_i [t.u.]
R_{a1}	5	2
R_{a2}	14	3
R_{a3}	23	3



ATTENZIONE SE RUBI AI PROCESSI SOTTO LA CAPACITA' (AS SERVER)!!
Il server, quando ruba capacità, è come se diventasse prioritario **COME** il processo alla quale la ruba!

Quindi, se RUBO capacità a P3 per gestire una richiesta asincrona, posso portare via l'esecuzione solo a P3 o a processi meno prioritari di lui (per esempio P4).

Se Rubassi a P1 (ma non è questo il caso in quanto è scarico) potrei togliere l'esecuzione (e runnare la gestione della richiesta asincrona) a tutti i processi compreso P1.

Qui vorrebbe eseguire P1, ruberei da P4 quindi non posso farlo in quanto P1 > P4.

Sporadic Server :gun:

Questa modalità è piuttosto complicata, quindi partiamo a spiegare da un esempio.
Dati questi processi e queste richieste asincrone:

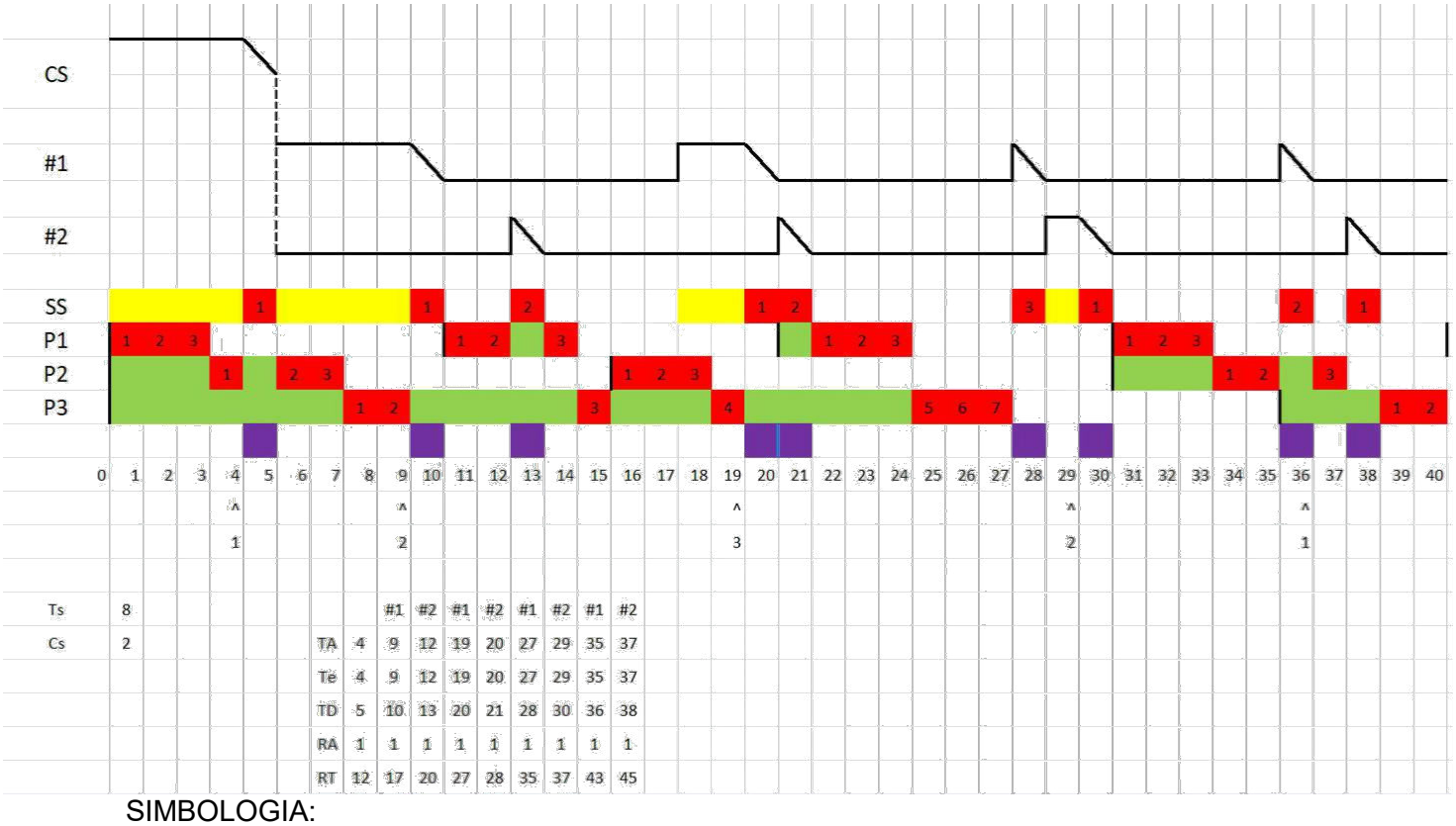
	T_i [t.u.]	C_i [t.u.]
P_1	10	3
P_2	15	3
P_3	35	7

	a_i [t.u.]	C_i [t.u.]
R_{a1}	4	1
R_{a2}	9	2
R_{a3}	19	3
R_{a4}	29	2
R_{a5}	36	1

Ed il periodo del server preso dal punto 2, è:

2. Server a priorità statica di periodo $T_s = 8$ t.u. e capacità $C_s = 2$ t.u. (strategia di schedulazione dei processi e del Server: RMPO)

La capacità può essere divisa in altri "sotto-servers" se non viene consumata tutta subito.
La caratteristica di questi sotto-servers è che si ricaricheranno in maniera asincrona tra di loro, permettendo una "ricarica media" più veloce.



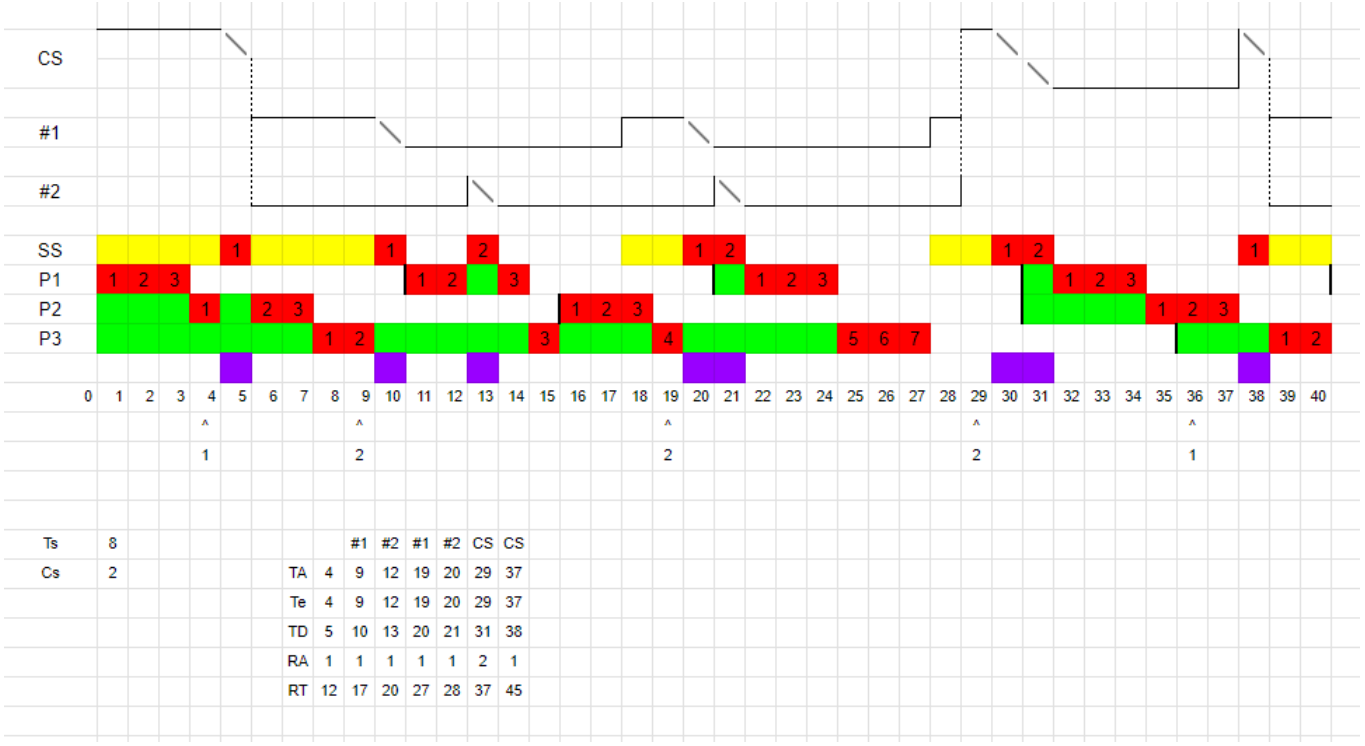
SIMBOLOGIA:

- **ROSSO** = Per rappresentare l'esecuzione sia di processi che di richieste asincrone
- **VERDE** = Se il processo vorrebbe eseguire ma non può eseguire perché altri di priorità superiore lo stanno già facendo
- **GIALLO**= SOLO per le richieste asincrone, quando il server ha capacità positiva (NB il server può anche essere verde se ho gente + prioritaria che esegue sopra)
- **BIANCO** = Quando ho terminato le mie esecuzioni e sono in attesa del prossimo periodo (per un processo) o di una successiva richiesta (per una Ra). Oppure quando il server ha capacità nulla
- **VIOLA** = il server è attivo: **la capacità del server è positiva** e **lui o un altro processo di priorità superiore sta eseguendo** (vedi esempio 3 con P1)

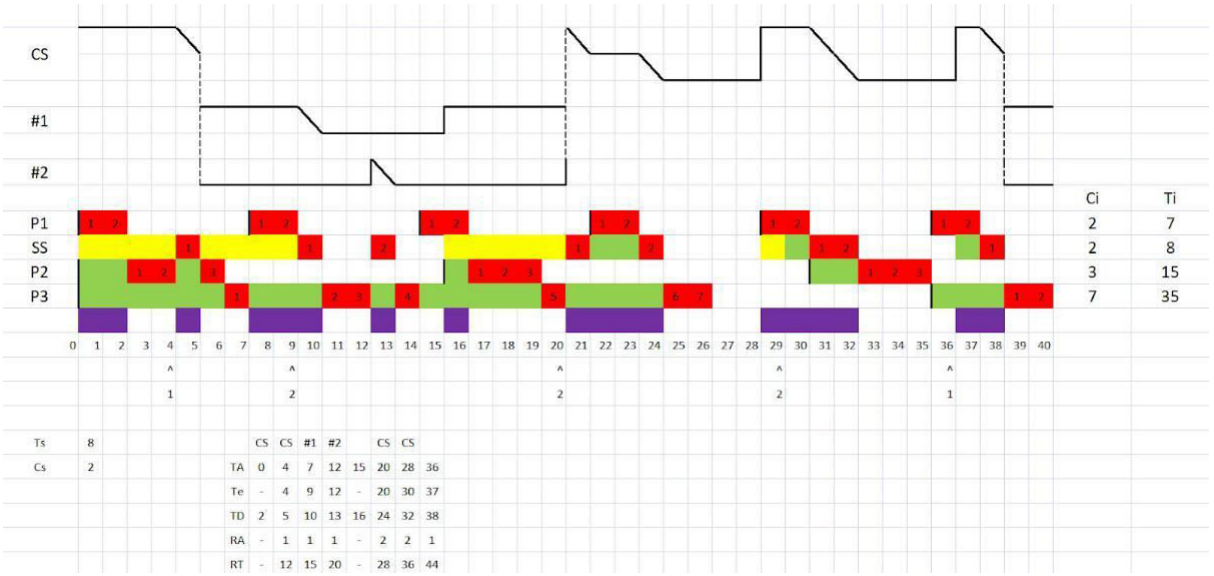
Questo esercizio deve essere svolto facendo in parallelo il **grafico colorato, il grafico della capacità C_S e la tabella.**

- La **divisione** avviene quando non ho consumato tutta la capacità durante la gestione di una richiesta. Le due pistole sono #1 carica (con il valore pari ad 1) ed #2 scarica. La #2 termina la ricarica all'istante RT calcolato nella tabella (Ovvero quando inizia l'ultimo viola incontrato + Ts del server).
- La **fusione** avviene appena entrambe le pistole possono sparare!
- Il server fa iniziare la ricarica quando **INIZIA UN VIOLA, se in quel tratto ho sparato.** Il tempo di ricarica coincide con Ts, quindi dopo Ts si torna full capacità.
- Calcolare un valore della tabella OGNI VOLTA che vedi c'è un nuovo gruppo di viola
- t_A è il tempo di inizio del server attivo (quando inizia il viola)
- t_E è primo istante in cui inizio a servire la richiesta aperiodica
- t_D è il tempo di fine del server attivo (quando finisce il viola)
- RA rappresenta quanto la capacità del server è calata durante l'esecuzione della richiesta aperiodica (quanti colpi hai sparato)
- $RT = max(t_A + T_S, t_D)$
- La capacità viene recuperata dal chunk senza capacità al tempo RT

In questo caso i chunk si riuniscono appena entrambi i server sono carichi:



Ts	8					#1	#2	#1	#2	CS	CS
Cs	2		TA	4	9	12	19	20	29	37	
			Te	4	9	12	19	20	29	37	
			TD	5	10	13	20	21	31	38	
			RA	1	1	1	1	1	2	1	
			RT	12	17	20	27	28	37	45	



Questo caso è più complicato, siccome i chunk si uniscono e alla fine si ri-dividono. Da notare la vicinanza dei colori **VERDE** e **GIALLO**. Questo è dovuto al fatto che la richiesta 4 arriva nello stesso momento in cui il server diventa attivo. Inoltre in questo caso il server NON è il più prioritario, bisogna quindi fare attenzione al colore **VIOLA** che ci sarà anche quando P1 è attivo (ed il server ha capacità > 0).
Notare inoltre come si ricarichi a 36, quando inizia effettivamente a eseguire le sue richieste a 30.

EDF

Total Bandwidth Server (TBS)

Ogni istante a_i la capacità del server assume un valore C_i ovvero il tempo di esecuzione della richiesta in arrivo.
Esegue il processo con deadline più vicina. Vanno quindi calcolati le deadline previste per i processi aperiodici.
Si calcola U_s come il fattore di utilizzazione del server. (Up = sommatoria dei Ci/Ti)

$$U_s = 1 - U_p$$

Devo eseguire per 3 volte? Ti fornisco 3 di capacità!

$$cs = Ci$$

Si aggiunge nella tabella dei processi una colonna delle deadline d_s calcolata come segue:

$$d_{si} = max(a_i,\ d_{s(i-1)}) + \frac{c_s}{U_s}$$

In caso il valore calcolato sia decimale si considera $d_s' = \lfloor d_s \rfloor$
(ovvero solo la parte intera: se calcolo 11,7 considero come deadline solo 11).
Nel grafico si esegue prima il processo (o la richiesta) con **deadline più imminente rivista in ogni istante di tempo.**

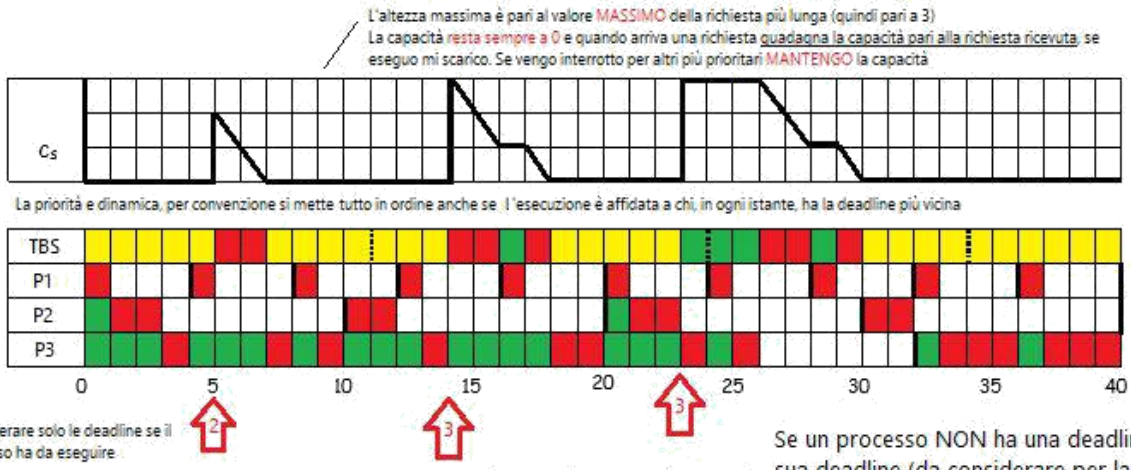
	T_i	C_i	U_i
P1	4	1	0.25
P2	10	2	0.2
P3	32	8	0.25

$$U_p = 0.25 + 0.2 + 0.25 = 0.7$$

$$U_s = 1 - U_p = 1 - 0.7 = 0.3$$

	a_i	C_i	d_{si}
Ra1	5	2	5 +(2 / 0.3) = 11.7
Ra2	14	3	max(11.7, 14) = 14 ⇒ 14 +(3 / 0.3) = 24
Ra3	23	3	max(23, 24) = 24 ⇒ 24 +(3 / 0.3) = 34

Le deadline sono quindi 11, 24 e 34. OCCORRE SEGNARLE SUL GRAFICO, ogni deadline rappresenta la deadline della richiesta i-esima. Quindi Ra1 avrà come scadenza 11, Ra2 come scadenza 14 ed Ra3 come scadenza 23. Tali deadline servono per scoprire chi deve eseguire (priorità dinamica) in quanto, per ogni istante, bisogna valutare chi esegue ovvero chi ha la deadline più vicina (Esegue TBS se è arrivata la richiesta e ovviamente la sua deadline è prima delle altre dei vari processi!).
Attento a non confondere le deadline di Ra1 con quelle di Ra2!!



Il server è SEMPRE giallo se il valore di Cs è pari a 0. Se ci sono tratti piani orizzontali, ovvero qualcuno mi ha rubato la priorità, allora è **verde**.

Considerare solo le deadline se il processo ha da eseguire

Deadline delle richieste calcolate con la formula:
Ra1 = 11
Ra2 = 24
Ra3 = 34
Segnati anche quanto durano le Ra e dove iniziano **con un tratteggio**

La priorità per una Ra in esame è da calcolare in base alla SUA deadline

	T_i	C_i
P1	4	1
P2	10	2
P3	32	8

Se un processo NON ha una deadline esplicita, la sua deadline (da considerare per la priorità dinamica) è pari al suo PROSSIMO periodo quindi D1 = 4,8,12,16... D2 = 10,20,30,40 D3 = 32,64
Ovviamente nel calcolo della priorità dinamica occorre considerare il processo in esame per "vincere" la gara di priorità SOLO SE ESSO NON HA ESEGUITO e vuole eseguire. Oppure puoi ignorarlo.

Solitamente nel TBS non vengono date le deadline dei processi, in quanto si assumono come quelle di default i loro Ti.! Per quanto riguarda le deadline delle richieste asincrone segnare quelle trovate nella tabella superiore con il calcolo!

NOTARE CHE QUI NON CONTA L'ORDINE CON CUI SCRIVO SERVER E PROCESSI NEL GRAFICO, LA PRIORITÀ DIPENDE DALLA DEADLINE!!
Per comodità si scrive sempre prima TBS :)
E' SENSATO SEGNARE LA DEADLINE DI OGNI RICHIESTA (SINGOLARMENTE) nel grafico per non confondersi!

Il server è sempre di colore **GIALLO**, se la sua capacità è positiva e non sta eseguendo perché altri processi di priorità superiore eseguono allora colore **VERDE** (Nei tratti piatti del grafico)

Constant Utilization Server (CUS) vs. TBS

PUNTO A)

Il CUS è un esercizio di confronto con il TBS.
1)Riempire la tabella bianca del prof con le stesse deadline per ogni richiesta asincrona calcolate nel punto precedente di TBS:

	a_i [t.u.]	C_i [t.u.]	d_{ai} [t.u.]
R_{a1}	6	2	6 + 2 / 0.346 = 11.8
R_{a2}	10	1	11.8 + 1 / 0.346 = 14.7
R_{a3}	14	4	14.7 + 4 / 0.346 = 26.2
R_{a4}	27	2	27 + 4 / 0.346 = 32.8

R_{a1}: $f_1(CUS)$ $f_1(TBS)$
R_{a2}: → $f_2(CUS)$ $f_2(TBS)$
R_{a3}: → $f_3(CUS)$ $f_3(TBS)$
R_{a4}: → $f_4(CUS)$ $f_4(TBS)$

2)Completare i campi delle varie Ra1...n scrivendo **ai : XXX**
al posto di XXX inserire il rispettivo tempo di arrivo (**ai**) della richiesta, reperibili dal testo d'esame (lasciando la prima Ra1 in bianco).
3)Indicare dopo la ai : xxx se tale valore è **> , < oppure =** alla deadline i-esima della tabella superiore.
4)In base al segno indicato nella sezione di sinistra delle Ra, abbiamo un corrispondente SEZIONE DESTRA da compilare con il segno corretto da inserire tra fi (CUS) **XXX** fi (TBS)

I valori a sinistra della Ra indicano che, al posto di XXX occorre inserire:

- Ra1 segnare sempre f1(CUS) = f1(TBS)
- ai < dsi e TBS è prioritario nell'intervallo $[a_i, ds_{i-1}]$, allora segnare fi(CUS) > fi(TBS)
- ai > dsi allora segnare fi(CUS) = fi (TBS)

Il prossimo passo richiede di completare una tabella che rappresenta il confronto tra i vari server:

Ai fini dell'esempio non si considera **BKG, PS e PES** ma solamente **TBS e CUS** in quanto i valori per i suddetti server sono reperibili dai loro rispettivi grafici.

- a_i e C_i = rispettivamente l'arrivo e la durata di tutte le richieste asincrone.
- f_i = istante in cui ho terminato di servire la richiesta asincrona considerata.
- $f_i - C_i - a_i$ = quantità di caselle in cui l'esecuzione è stata bloccata

COLONNA TBS
1)fi = Quando ho terminato di servire la richiesta asincrona considerata
2)fi-ai-Ci = semplice sottrazione tra fi, il tempo di arrivo della richiesta e il Ci della richiesta.

COLONNA CUS
1)Se fi (CUS) = fi (TBS) inserire sia in fi che in fi-ai-Ci un trattino "-"
2)Se fi (CUS) > fi(TBS) occorre calcolare il valore con la **formula sotto***

Formula:

$$f_i(CUS) = f_i(TBS) + ds_{i-1} - a_i \; - < \textit{tempo in ready del server nell'intervallo } [a_i; ds_{i-1}] \textit{ in TBS} > \forall\, i$$

questa è un'ulteriore verifica per i segni = o < messi in precedenza al punto 3.

Esempio applicazione formula nel caso Ra3:
I due valori da confrontare sono 14 e 14.7, la formula applicata diventa:

$f_3(CUS) = 21 + 14.7 - 14 - 0 = 21.7$

Togliere il numero di volte in cui il TBS è ready nell'ultimo pezzo della formula, in questo caso NON è in ready quindi − 0 .

R_{a1}: $f_1(CUS) = f_1(TBS)$
R_{a2}: $a_2 = 10 < d_{s1} = 11.8 \rightarrow f_2(CUS) > f_2(TBS)$
R_{a3}: $a_3 = 14 < d_{s2} = 14.7 \rightarrow f_3(CUS) > f_3(TBS)$
R_{a4}: $a_4 = 27 > d_{s3} = 26.2 \rightarrow f_4(CUS) = f_4(TBS)$

	a_i	C_i	f_i	$f_i-a_i-C_i$	f_i	$f_i-a_i-C_i$	f_i	$f_i-a_i-C_i$	f_i	$f_i-a_i-C_i$	f_i	$f_i-a_i-C_i$	f_i	$f_i-a_i-C_i$
R_{a1}	6	2	21	13	12	4	11	3	11	3	8	0		
R_{a2}	10	1	22	11	14	3	12	1	12	1	11	0	12.8	1.8
R_{a3}	14	4	29	11	32	14	21	3	28	10	21	3	21.7	3.7
R_{a4}	27	2	32	3	36	7	29	0	30	1	29	0		
			BKG		PS		PES		SS		TBS		CUS	

Vediamo questo **caso particolare:**

	a_i [t.u.]	C_i [t.u.]	d_{si} [t.u.]
R_{a1}	5	2	5 + 2 / 0.3 = 11.7
R_{a2}	14	3	14 + 3 / 0.3 = 24
R_{a3}	23	3	24 + 3 / 0.3 = 34

	a_i	C_i	f_i	$f_i-a_i-C_i$	f_i	$f_i-a_i-C_i$	f_i	$f_i-a_i-C_i$	f_i	$f_i-a_i-C_i$	f_i	$f_i-a_i-C_i$
R_{a1}	5	2	19	12	11	4	7	0	7	0	-	-
R_{a2}	14	3	26	7	26	7	18	1	18	1	-	-
R_{a3}	23	3	30	4	35	9	27	1	30	4	-	-
			BKG		PS		PES		TBS		CUS	

R_{a1}: $f_1(CUS) = f_1(TBS)$
R_{a2}: $a_2 = 14 > d_{s1} = 11.7$ $\rightarrow f_2(CUS) = f_2(TBS)$
R_{a3}: $a_3 = 23 < d_{s2} = 24$ TBS non è prioritario in [23-24] $\rightarrow f_3(CUS) = f_3(TBS)$

Il server in [23-24] è in ready, quindi applicando la formula di sopra avremmo:

$f_3(CUS)=30+24-23-1=30+1-1=30$

Siccome il risultato ottenuto è uguale alla $f_3(TBS)$ possiamo scrivere "nell'intervallo [23-24] TBS non è prioritario".

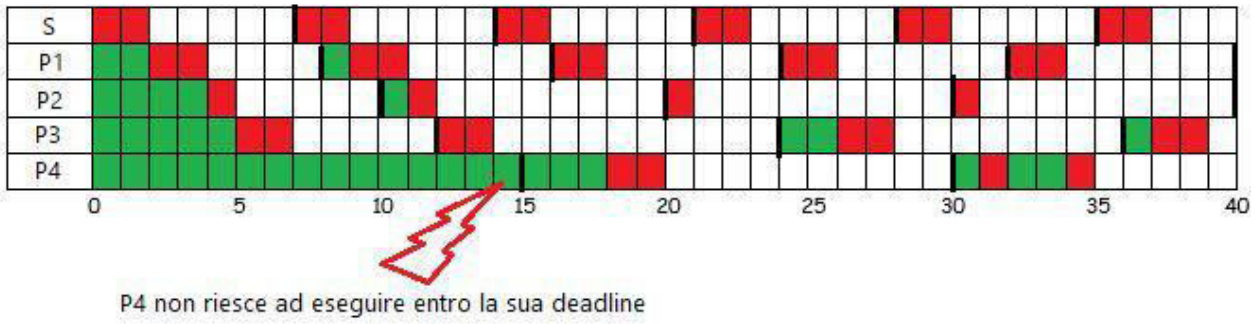
PUNTO B)

a) caso processi P1, P2, P3

Si completa la tabella utilizzando RMPO ricordandosi di elencare i processi (+ il server S) in ordine crescente di T_i .

	T_i [t.u.]	C_i [t.u.]
P_1	8	2
P_2	10	1
P_3	12	2
P_4	15	2

Ti del server = 7
Cs del server = 2



P4 non riesce ad eseguire entro la sua deadline

La missed deadline avviene se, dopo aver fatto RMPO (senza considerare le richieste asincrone), uno qualsiasi dei processi non riesce a completare la sua esecuzione entro la sua deadline. (Ricorda, se non viene indicata la deadline è uguale al periodo T_i)

Se non ho missed-deadline ⇒ Risposta NO, Cs ≤ Cs dato

Se ho almeno una missed-deadline ⇒ Risposta SÌ ⇒ $C_S \leq T_1 - T_s$ (se server prioritario)
Nell'esempio la risposta è Sì in quanto ho una missed deadline all'istante 14.

b) caso insieme di processi P1', P2' P3' con stessi Ui e Ti
(NB Up = somma di tutti i Ui ovvero Ci/Ti)

$$U_s = C_s / T_s \qquad U_{s\,max} = \frac{2}{(1 + \frac{U_p}{N})^N} - 1,$$

Con N = numero di processi tranne il server

Controlliamo che $U_{s\,max} \geq U_s$
Se sì la risposta è NO, viceversa è SI

Infine, con il Ts del server (testo), calcoliamo:

$$C_{s\,max} = T_s \bullet U_{s\,max}$$

Nell'esempio:

$$U_s = 2 / 7 = 0.286 \qquad U_p = 0.65$$

$$U_{s\,max} = \frac{2}{(1 + \frac{U_p}{N})^N} - 1 = \frac{2}{(1 + \frac{0.65}{4})^4} - 1 = 0.095$$

$$C_{s\,max} = T_s \bullet U_{s\,max} = 7 \bullet 0.095 = 0.665$$

$$U_s = 0.286 > U_{s\,max} = 0.095 \Rightarrow \text{SI}$$

PROBLEMA 3

Legenda simboli:

- Le **X** si segnano sopra quando è in esecuzione un processo inferiore(cioè di sotto).
- I **O** si segnano sui processi che “vengono saltati” purché siano ready (sfondo verde), partire dal processo che ha ∇ / □ fino a quello che esegue (ovviamente escluso)
- I ∇ / □ si segnano quando cedo la priorità, ovvero quando vengono sospeso (e cedo la priorità)
  ∇ si usa per PCP, IPCP e SRP
  □ si usa per PIP

DMPO

Sono dati un insieme di processi con i rispettivi ϕ_i indicanti gli istanti di tempo di partenza di tali processi:

	T_i [t.u.]	D_i [t.u.]	C_i [t.u.]
P_1	20	20	5
P_2	25	25	6
P_3	35	30	6
P_4	45	45	3
P_5	65	60	6

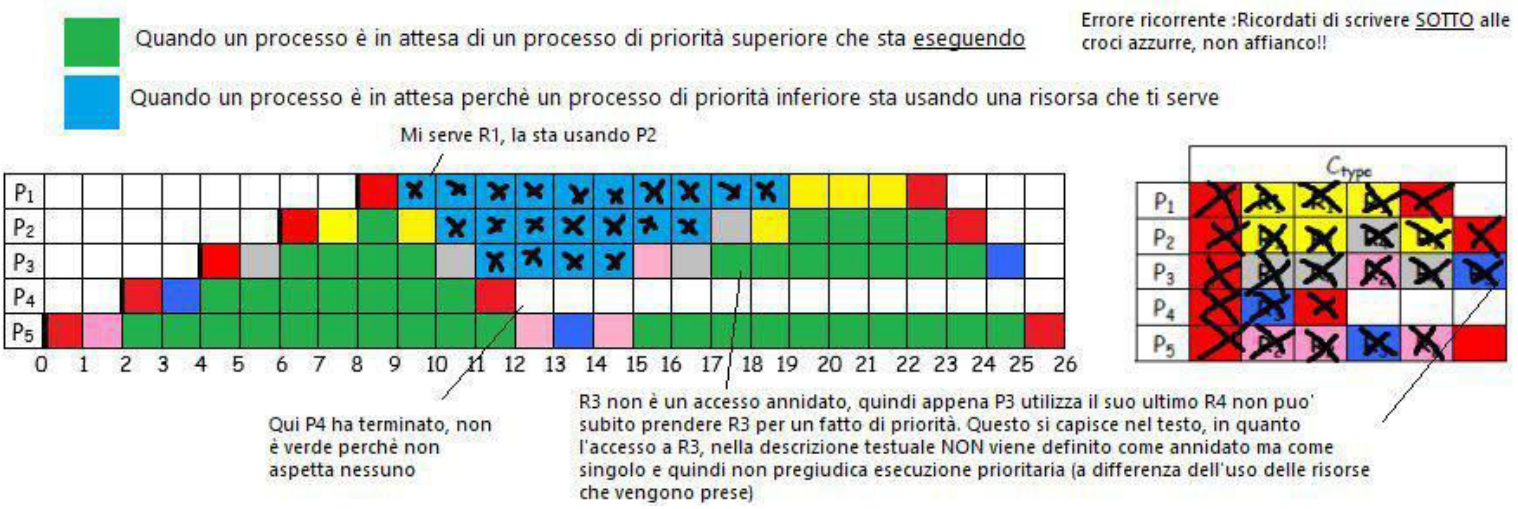
attivati all'istante ϕ_i ([t.u.]) = 8, 6, 4, 2, 0, rispettivamente, è affidata ad un sistema di elaborazione monoprocessore.

Questi processi condividono un certo numero di risorse indicate in questa tabella (NB: controllare che nella descrizione testuale gli accessi siano annidati anche se solitamente è SEMPRE così)

	C_{type}					
P_1		R_1	R_1	R_1		
P_2		R_1	R_1	R_4	R_1	
P_3		R_4	R_4	R_2	R_4	R_3
P_4		R_3				
P_5		R_2	R_2	R_3	R_2	

NOP

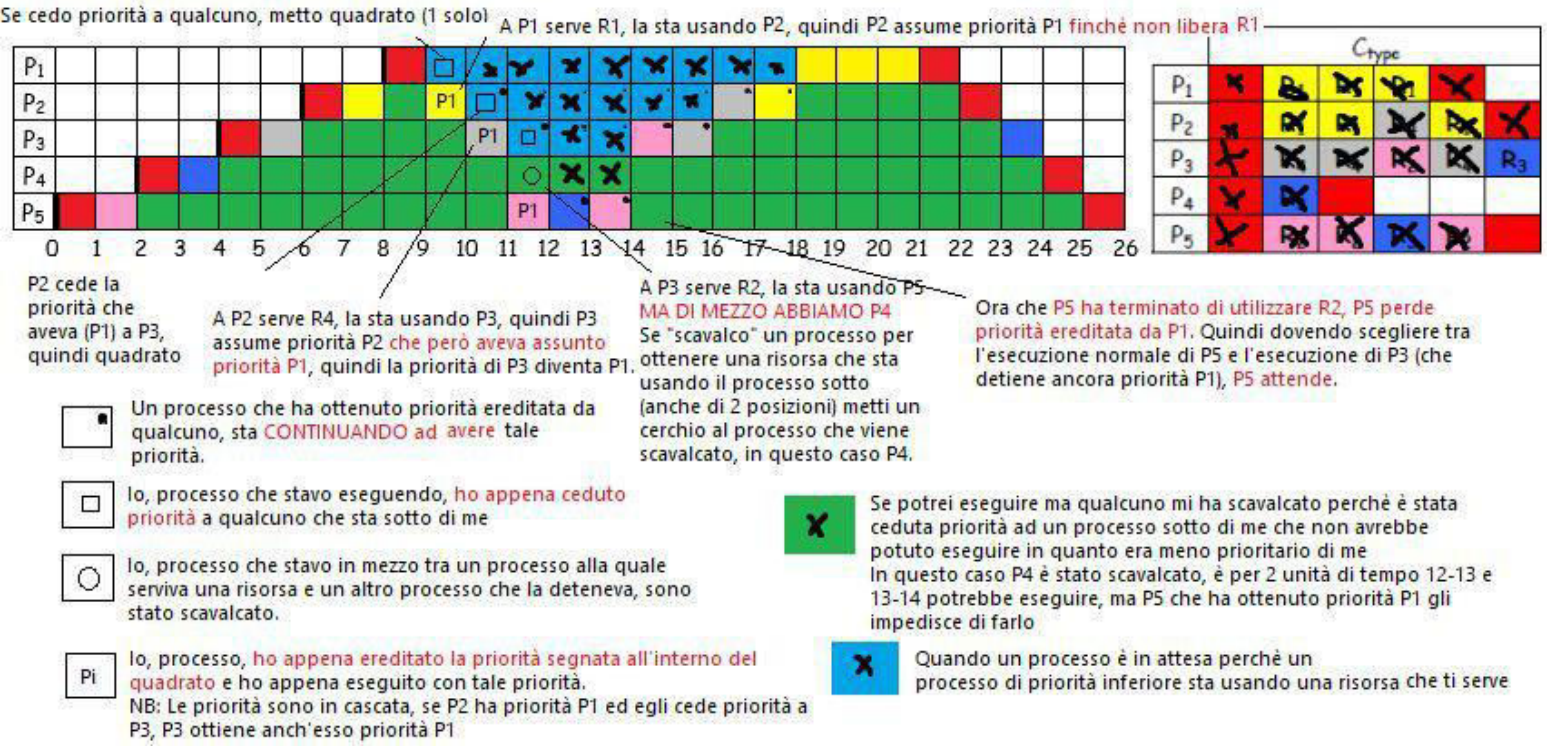
POSSO ESEGUIRE? NO PERCHÉ LA RISORSA LA STA USANDO QUALCUN ALTRO, QUINDI NON FACCIO NIENTE E STO FERMO, ESEGUE QUELLO CON PRIORITA SUBITO INFERIORE.



- Comincia a fare eseguire in ordine di precedenza normale con Priorità P1 > priorità P5 ma partendo verso il basso in quanto i processi partono in base alla loro ϕ_i , quindi nell'esempio il primo a partire è P5, poi verrà P4, poi P3 etc etc…
- NB: Se un processo inizia con il colore AZZURRO, rimane AZZURRO finchè non può nuovamente eseguire!!
- Attenzione agli accessi annidati!
  Nell'istante **[17-18] P3 non può eseguire usando la risorsa blu R3 in quanto l'accesso annidato è solamente di 4 elementi.**
  X X X X
  All'istante successivo, P3 non può eseguire X in quanto liberando X è meno prioritario di P2 che la stava attendendo!

PIP

POSSO ESEGUIRE? NO PERCHÉ LA RISORSA LA STA USANDO QUALCUN ALTRO, QUINDI NON FACCIO NIENTE E STO FERMO, ESEGUE CHI HA LA RISORSA CHE MI SERVIVA PER ESEGUIRE(e fai quadrato perchè cedi la tua priorità a lui).



Comincia a fare eseguire in ordine di precedenza normale con Priorità P1 > priorità P5 ma partendo verso il basso in quanto i processi partono in base alla loro ϕ_i , quindi nell'esempio il primo a partire è P5, poi verrà P4, poi P3 etc etc…

TUTTI I PROCESSI CHE VENGONO SCAVALCATI DA QUALCUNO CON IL QUADRATO, SONO VERDI COL CERCHIO.

Se un processo B ha ereditato priorità da A ed è presente un processo C che eredita da B, allora C prende la priorità di A come nel caso di P3 nell'esempio.

QUI NON CI SONO TRIANGOLI

Le priorità che eredita un processo si impilano in una LIFO. Ogni priorità viene restituita quando rilascio la risorsa associata.
Appena restituisco una priorità, riprendo immediatamente quella subito sotto nello stack, anche se sono in blocco.

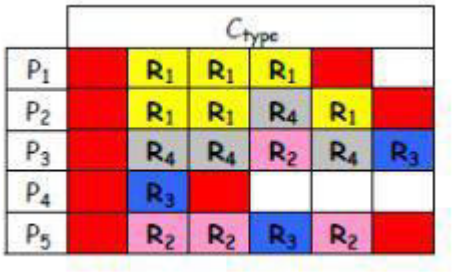
PCP

POSSO ACQUISIRE UNA RISORSA? LA RISPOSTA ALLA DOMANDA PER OGNI PROCESSO DIPENDE DAL LIVELLO ATTUALE DEL PRIORITY CEILING (IL VALORE). Solo processi di priorità STRETTAMENTE MAGGIORE DEL valore indicato nel PRIORITY CEILING POSSONO ottenere tale risorse, oppure si bloccano (SPESSO CEDENDO IL PROPRIO LIVELLO DI PRIORITÀ AD UNO QUALSIASI DEI PROCESSI IN BASSO ANCHE NON ADIACENTI)

SE RILASCIO UNA RISORSA IL PRIORITY CEILING TORNA AL LIVELLO PRECEDENTE. [Il grafico è considerato solo durante le ACQUISIZIONI delle risorse, durante esecuzioni normali se un processo può eseguire, esegue.]

I processi eseguono in esecuzione normale considerando le loro ϕ_i

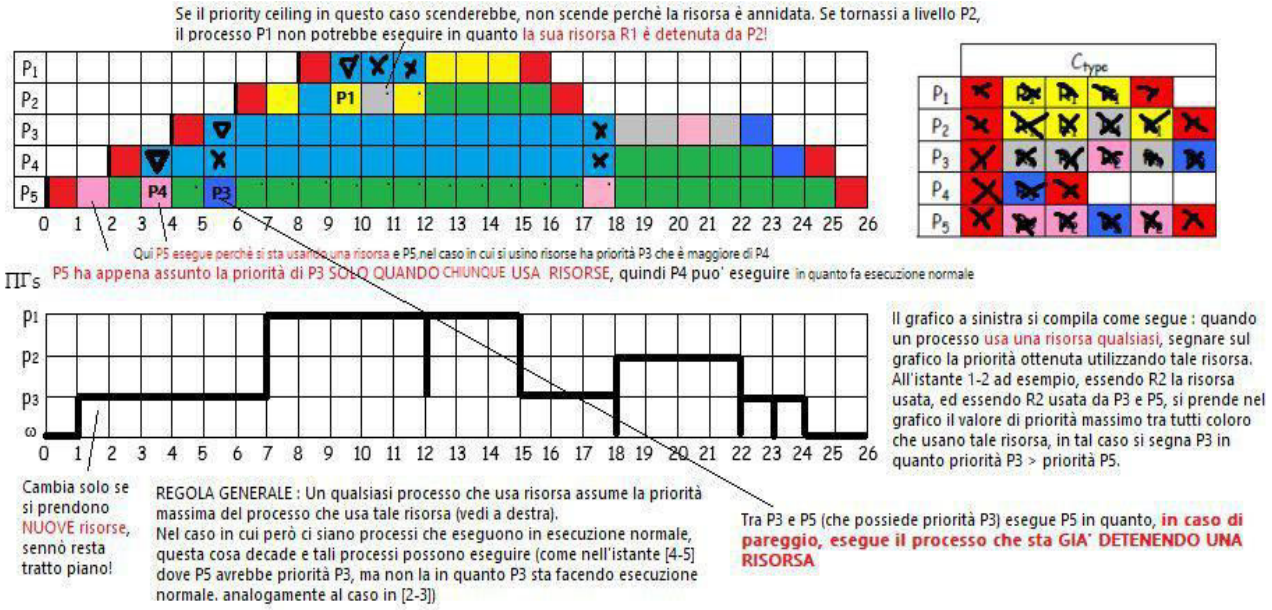
Si comincia analizzando la tabella delle risorse, segnandosi il processo che usa la risorsa i-esima in base alla priorità del processo PIÙ ALTO che la sta usando.



Ad esempio R1 è usata da P1 e P2 quindi il processo massimo che la usa ha priorità P1.
Processi più prioritari per ogni risorsa:
R1 -> P1
R2 -> P3
R3 -> P3
R4 -> P2

- Ogni volta che
  1)Una NUOVA risorsa viene presa da un processo viene modificato il priority cealing al valore corrispondente calcolato sopra.
  2)Una risorsa viene rilasciata da un processo: il priority cealing ritorna al valore precedentemente imposto o a zero.
- **Quando un processo stabilisce un priority cealing acquisendo una nuova risorsa NON può <u>auto-bloccarsi</u>, egli continuerà a correre a meno che un processo non batta il priority cealing che ha appena impostato.**
- **Se un processo, impostando il priority cealing, blocco dei processi soprastanti, il processo corrente EREDITA LA PRIORITÀ DEL PROCESSO BLOCCATO.**

SIMBOLOGIA: Valgono le regole degli altri esercizi, con l'aggiunta dei triangoli.
Inserire i triangolini con la punta verso il basso <u>solo una volta</u> (seguito da tante croci verdi finchè sono bloccato) nei momenti in cui il processo considerato NON ha potuto eseguire a causa di un qualsiasi processo che si trova sotto di lui, che ha ottenuto potere prioritario rispetto al processo considerato.



ES: Se sono P4 e batterei P5 durante la normalità, ma P5 ha ottenuto priorità P1, P2 oppure P3 per qualche motivo, allora metto il triangolino su P4 perché non può eseguire e continuo a croccettare finché P5 non ritorna a priorità inferiore alla mia (dopo il rilascio della risorsa)

NB: In tutti i diagrammi “a onde” delle priorità (anche i seguenti), a meno di risorse annidate, il livello di priorità scende sempre quando rilascio una risorsa, eventualmente salendo immediatamente se ne acquisisco un’altra (se quest’ultima ha priorità superiore). In caso di risorse annidate non scende mai, può però eventualmente salire se la risorsa annidata ha priorità superiore (scenderà al livello precedente quando viene rilasciata)

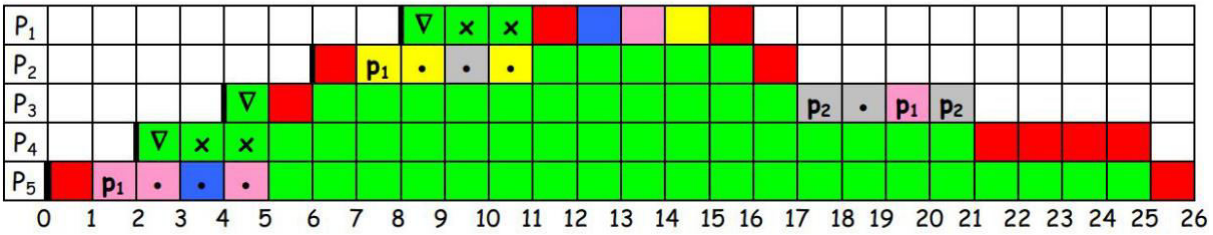
IPCP

Uguale a PCP, unica differenza:

- Non appena un processo acquisisce una risorsa, questo assume priorità pari a quella del processo di priorità massima che utilizzerebbe tale risorsa (anche se il processo di priorità superiore non è partito o ha terminato).
- Nel grafico non ci sono **BLU**, solo **VERDI**.

	T_i [t.u.]	D_i [t.u.]	C_i [t.u.]
P_1	20	20	5
P_2	25	25	6
P_3	35	30	5
P_4	45	45	4
P_5	50	50	6

	C_{type}				
P_1		R_3	R_2	R_1	
P_2		R_1	R_1	R_4	R_1
P_3		R_4	R_4	R_2	R_4
P_4					
P_5		R_2	R_2	R_3	R_2



Massimo tempo di blocco (PIP e PCP)

- PIP -> B_i = *min*(∑ *max*(*ogni riga sotto*), ∑ *max*(*ogni colonna a partire dalla riga sotto*))
- PCP/IPCP -> B_i = *max*(*processi sottostanti* [== *righe sotto*])

Tabella da completare verticalmente:

➔ Quante volte P1 usa R1? 0 quindi lascio vuoto
➔ Quante volte P1 usa R2? 1
➔ Quante volte P3 usa R2? 3 perché R2R2R2 è accesso annidato.
➔ Quante volte P3 usa R1? 6 perchè devi considerare gli accessi annidati ad R1.

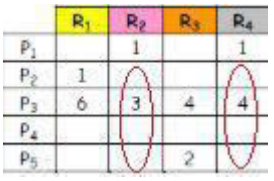
Una volta fatta la tabella, occorre calcolare i Bi.
Si prende la tabella e si calcolano i Bi come segue:

	C_{type}														
P_1		R_2	R_4												
P_2		R_1													
P_3	R_1	R_4	R_4	R_4	R_4	R_1		R_2	R_4	R_2		R_3	R_4	R_4	R_3
P_4															
P_5		R_3	R_3												

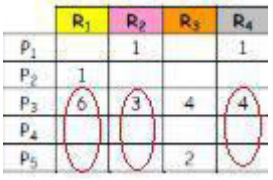
	R_1	R_2	R_3	R_4	B_i	
P_1		1		1	4	4
P_2	1				6	6
P_3	6	3	4	4	2	2
P_4					2	2
P_5			2		0	0
					PIP	PCP

Colonna ***PCP***

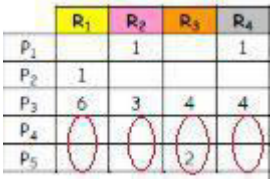
1. Guardare la tabella affianco ai Bi costruita precedentemente.
   Iniziamo con P1 NON considerando la sua riga (cancellandola mentalmente) ai fini del calcolo del valore Bi della tabella.
   Si considerano **solamente tutte le colonne sottostanti** in cui P1 usa almeno 1 risorsa (in questo caso solo R2 ed R4 in quanto gli altri spazi sono vuoti).
   Il valore da segnare in tabella è semplicemente il **MASSIMO** tra tutti i numeri presenti nelle colonne (verticalmente) "sbloccate" dal processo. (Max (1,3,1,4) = 4)



2. L'arrivo di un nuovo processo può sbloccare ulteriori colonne nella considerazione del massimo. P2, oltre a far cancellare mentalmente la sua riga e di tutti quelli sopra (P1), prende in considerazione la risorsa R1 in quanto la utilizza. A questo punto per il calcolo del massimo avremo: Max(1,6,1,3,1,4) = 6) in quanto la colonna di R1 è stata sbloccata.



3. L'arrivo di P3 sblocca **tutte le restanti colonne** in quanto utilizza tutte le risorse, viene cancellata la sua riga e quelle soprastanti e si resta semplicemente con la riga bianca di P4 e un 2 (della riga di P5), il risultato è quindi solo 2.



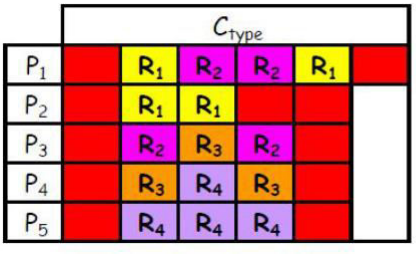
4. Analogamente per P4 avremo lo stesso risultato, mentre per P5, non avendo numeri, scriveremo 0.

N.B.: ricordarsi che vogliamo il massimo tempo di blocco, cioè il caso peggiore

Colonna ***PIP***

1. Considero le colonne delle risorse usate dal processo e da quelli superiori. (come PCP)
2. Prendendo al massimo un valore per riga (inferiori a quella del processo in considerazione) e uno per colonna (non devo avere due valori sulla stessa riga o sulla stessa colonna), prendo quelli che mi danno la somma maggiore possibile.
   Nell'esempio ho sempre valori su una sola riga, per cui non sommo mai niente.
3. In caso di risorse annidate "sblocco" anche la colonna della risorsa annidata

CASO PARTICOLARE Risorse annidate:



	R1	R2	R3	R4
P1	4	2		
P2	2			
P3		3	1	
P4			3	1
P5				3

P1 = 11
P2 = 9
P3 = 6
P4 = 3
P5 = 0

Audsley (the return)

Seguire quanto visto nell'algoritmo di Audsley del PROBLEMA 1 considerando però come unica differenza il calcolo della R0 iniziale.

$$R^0_i = C_i + max\{B_{i\ P\ CP}\ ,\ B_{i\ P\ IP}\}$$

Il valore della R0 per ogni processo è calcolato con la formula sopra. Nel nostro esempio e, considerando i BI sia di PCP che di PIP calcolati nel punto precedente avremo:

	T_i [t.u.]	D_i [t.u.]	C_i [t.u.]
P_1	20	20	5
P_2	25	25	6
P_3	35	30	6
P_4	45	45	3
P_5	65	60	6

$R^0_1 = 5 + max(13, 4) = 18$
$R^0_2 = 6 + max(9, 4) = 15$
$R^0_1 = 6 + max(5, 4) = 11$
$R^0_1 = 3 + max(4, 4) = 7$
$R^0_1 = 6 + max(0, 0) = 6$

Si procede con nel modo classico.
Verificare alla fine se la Ri di Audsley è <= al Di del processo.
Se c'è almeno un no la risposta è sempre conclusiva **NN/SS**

EDF

SRP

	T_i [t.u.]	D_i [t.u.]	C_i [t.u.]
P_1	15	15	3
P_2	20	20	3
P_3	25	23	3
P_4	30	26	6
P_5	35	29	6

ϕ_i ([t.u.]) = 7, 5, 4, 2, 0,

	C_{type}					
P_1		R_1				
P_2		R_2				
P_3						
P_4		R_1	R_1	R_2	R_2	R_1
P_5		R_2	R_2	R_2	R_2	

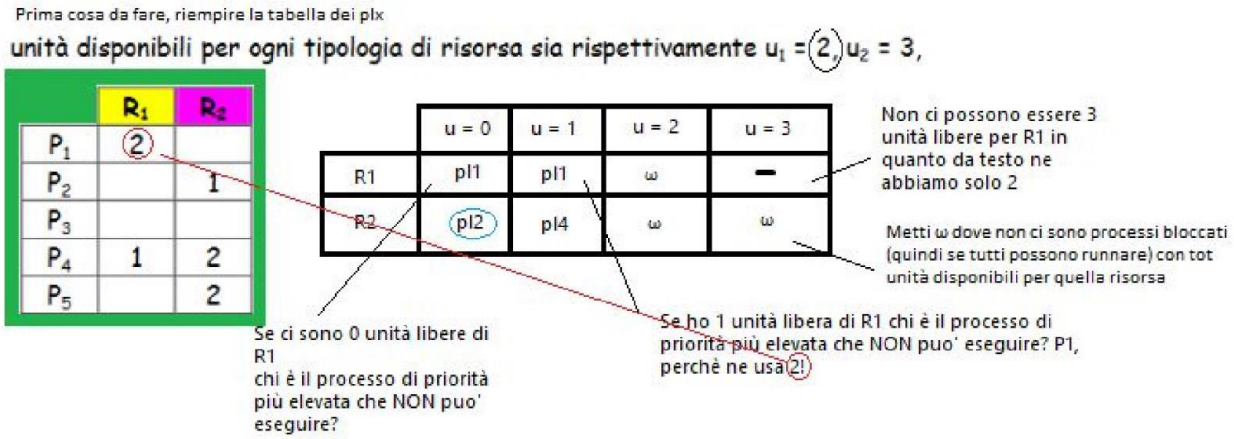
unità disponibili per ogni tipologia di risorsa sia rispettivamente $u_1 = 2$, $u_2 = 3$,

	R_1	R_2
P_1	2	
P_2		1
P_3		
P_4	1	2
P_5		2

L'ultima tabella indica il numero di unità necessarie di quella risorsa richieste dal processo (P1 ha bisogno di 2 unità R1).

Le esecuzioni di qualsiasi processo o l'acquisizione di nuove risorse può avvenire SOLAMENTE se il livello di priorità del processo è STRETTAMENTE MAGGIORE al livello di priorità del grafico sottostante. in tutti gli altri casi mi sospendo

- Riempire una nuova tabella ponendosi questa domanda: “**Chi è il processo di priorità maggiore che non può eseguire se ho disponibili u = x unità?**”
    - Se, per esempio u =1 per la colonna R2 è P4, scrivere pl4 in quanto processo massimo che non può eseguire (P4 userebbe 2 unità, e ne abbiamo solo 1).
    - Se tutti possono eseguire, scrivere ω (che indica la priorità minima del sistema, più bassa anche di quella dell’ultimo processo)
- La tabella indicherà per ogni unità disponibile di una risorsa, il tetto di priorità
-

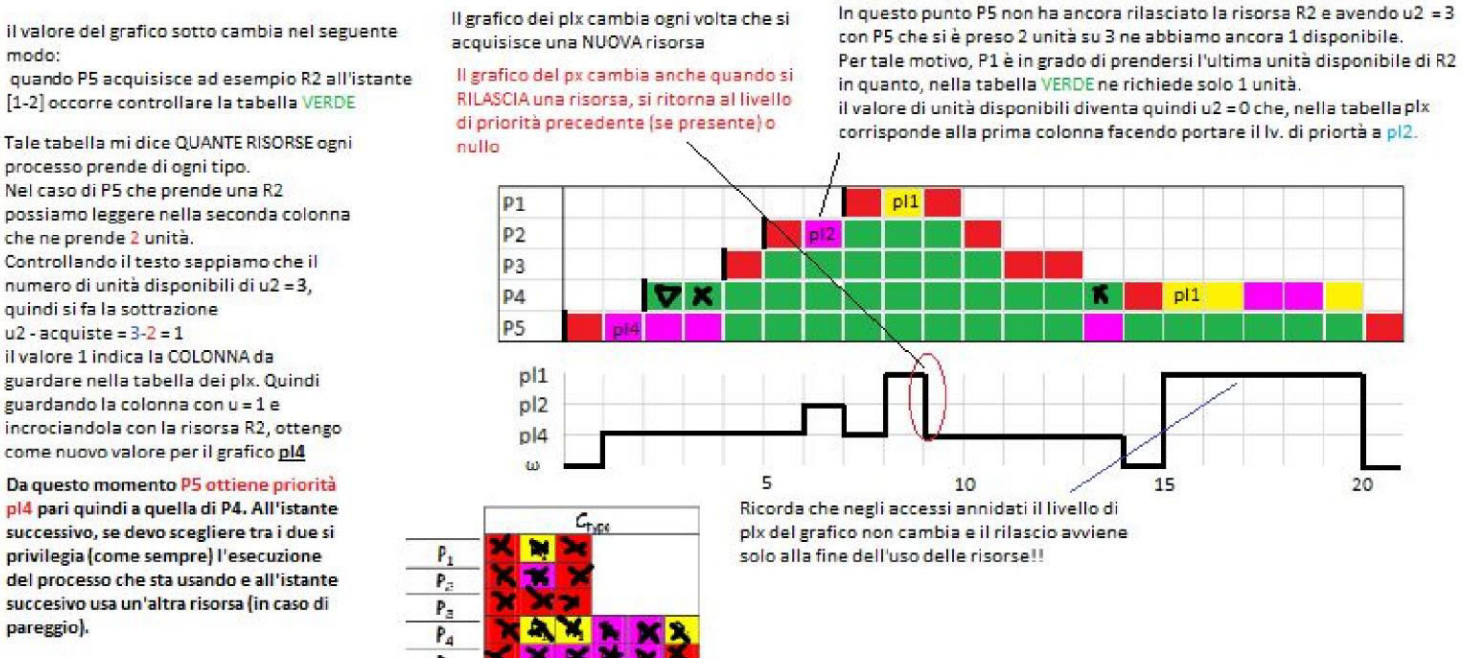


Disegnare il grafico:

- Se una qualsiasi risorsa acquisita farebbe SCENDERE il lv di priorità, questo non succede MAI in quanto il tetto di priorità scende solamente durante un RILASCIO. Anche in questo caso si prende quindi il MASSIMO a parità di acquisizioni.
- Ogni volta che si acquisisce una risorsa, cambia il tetto di priorità.
  Se P5 utilizza 2 unità di R2 e ce ne sono disponibili 3, si deve guardare la colonna u = 1 (rimane solo un’unità disponibile ora) della tabella dei livelli di priorità per sapere come cresce il grafico.
- Anche qui a parità di priorità, esegue il processo che detiene la risorsa ed è in esecuzione
- Al rilascio di una risorsa, cambia il livello di priorità nel grafico plx che torna al lv prec.

NB: In questo esercizio compare il simbolo del ∇.

Esso è da inserire ogni volta che un processo NON ha potuto eseguire a causa del fatto che sua priorità era INFERIORE al livello di priorità del grafico. Come accade per P4 nell’esempio di sopra.



Questa tabella è la stessa di TBS
NB: L'ultima colonna è il numero di verdi per ogni processo.

f_i rappresenta ogni istante in cui un processo termina con la sua ultima cosa da fare
Ci sono quelli del testo

ϕ_i viene data dal testo, è lo sfasamento di partenza (ritardo) per ogni processo.

	ϕ_i	C_i	f_i	$f_i - \phi_i - C_i$
P_1	7	3	10	0
P_2	5	3	11	3
P_3	4	3	13	6
P_4	2	6	20	12
P_5	0	6	21	15
			SRP	

C'è sempre la solita tabella con i f_i .

Calcolo del massimo tempo di blocco (SRP)

Qui occorre calcolare i B_i che però sono spesso gli stessi di PCP/IPCP:

$$B_i = max \{Z_{jk}: pl_j < pl_i \wedge pl_j < \Pi\Lambda_k(0) \} \forall i$$

A parole: il massimo tempo di utilizzo di una risorsa (Z_{jk}) purché:

1) tale valore sia riferito a un processo con priorità plj inferiore al processo corrente
2) la priorità del processo plj sia inferiore alla priorità che la risorsa k assume se $u_k = 0$

Ovvero:
Prendo il valore più grande che trovo tra la tabella a patto che:

1) sia in una riga più bassa di quella del processo preso in considerazione
2) il processo Pj relativo abbia priorità inferiore a quella relativa alla risorsa con $u_k = 0$ (prima colonna della tabella compilata all'inizio di SRP)

Consiglio: segnarsi prima del calcolo le priorità relative a ogni risorsa per $u_k = 0$.

	R_1	R_2	B_i
P_1	1		5
P_2		1	5
P_3			5
P_4	5	2	4
P_5		4	0

Analisi di schedulabilità

$$\forall i \in P,\ \sum_{k=1}^{i} \left(\frac{C_k}{D_k}\right) + \frac{B_i}{D_i} \leq 1$$

NB: I valori per ogni processo vanno da 1 fino al processo considerato.

	R_1	R_2	B_i
P_1	1		5
P_2		1	5
P_3			5
P_4	5	2	4
P_5		4	0

	T_i [t.u.]	D_i [t.u.]	C_i [t.u.]
P_1	15	15	3
P_2	20	20	3
P_3	25	23	3
P_4	30	26	6
P_5	35	29	6

Per P4 ad esempio ci sono i termini di P1, P2, P3 e P4

P1: $C_1 / D_1 + B_1 / D_1 = 3/15 + 5/15 = 0.53 \leq 1$

P2: $C_1 / D_1 + C_2 / D_2 + B_2 / D_2 = 3/15 + 3/20 + 5/20 = 0.8 \leq 1$

P3: $C_1 / D_1 + C_2 / D_2 + C_3 / D_3 + B_3 / D_3 = 3/15 + 3/20 + 3/23 + 5/23 = 0.7 \leq 1$

P4:

$C_1 / D_1 + C_2 / D_2 + C_3 / D_3 + C_4 / D_4 + B_4 / D_4 = 3/15 + 3/20 + 3/23 + 6/26 + 4/26 = 0.865 \leq 1$

P5:

$C_1 / D_1 + C_2 / D_2 + C_3 / D_3 + C_4 / D_4 + C_5 / D_5 + B_5 / D_5 = 0.918 \leq 1$

NB: B5/D5 = 0/29 = 0

In questo caso sono tutti minori o uguali di 1 quindi ***SS***

NB: basta un solo No per far fallire il test con NN in caso contrario inserire SS.

***Nota sulle risorse annidate generali:**
In qualsiasi esercizio viene spesso chiesto di aggiornare grafici durante l'acquisizione/rilascio di risorse.
In presenza di **Risorse annidate** fa cambiare le cose:
ES risorsa annidata aquisita
P2 -> R1 R2 R1
In questo caso, un processo più prioritario **P1 dovrà**
1)Se vuole R1ESTERNA deve aspettare PER FORZA LA FINE DI TUTTEle operazioni di P2
2)Se vuole la R2 INTERNA può attendere fino all'istante 2, dopodichè può subentrare e non attendere l'esecuzione della R1 in quanto più prioritario.

Il lock di una risorsa arriva dal PRIMO uso della risorsa stessa!
Nel caso di risorse annidate ci sono SEMPRE **due acquisizioni e due rilasci** per le risorse coinvolte.
P2 acquisisce R1 poi acquisisce R2, rilascia R2 e infine rilascia R1. Per questo motivo P1 può prendersi la R2 subito dopo il rilascio di P2!

Possono essere anche multiple come nel caso R1 R1 R2 R2 R1

Attenzione a non confondersi con il TETTO di priorità, perché in alcuni grafici si deve prendere il MASSIMO derivante dall'operazione annidata, ed è per questo che il grafico non viene aggiornato in quei casi!
Se per esempio in un esercizio che chiede di fare il priority level abbiamo
R1 -> priorità 2
R2 -> priorità 3
Il grafico NON si aggiornerebbe ma resterebbe a priorità 2 in quanto la priorità di R2 è inferiore a quella di R1!